TOSHIBA

TX03 Peripheral Driver Usage Example (TMPM384)

Ver 1

Sep, 2017

TOSHIBA ELECTRONIC DEVICES & STORAGE CORPORATION

CMDR-M384UE-01E

RESTRICTIONS ON PRODUCT USE

- DO NOT USE THIS SOFTWARE WITHOUT THE SOFTWARE LISENCE AGREEMENT.

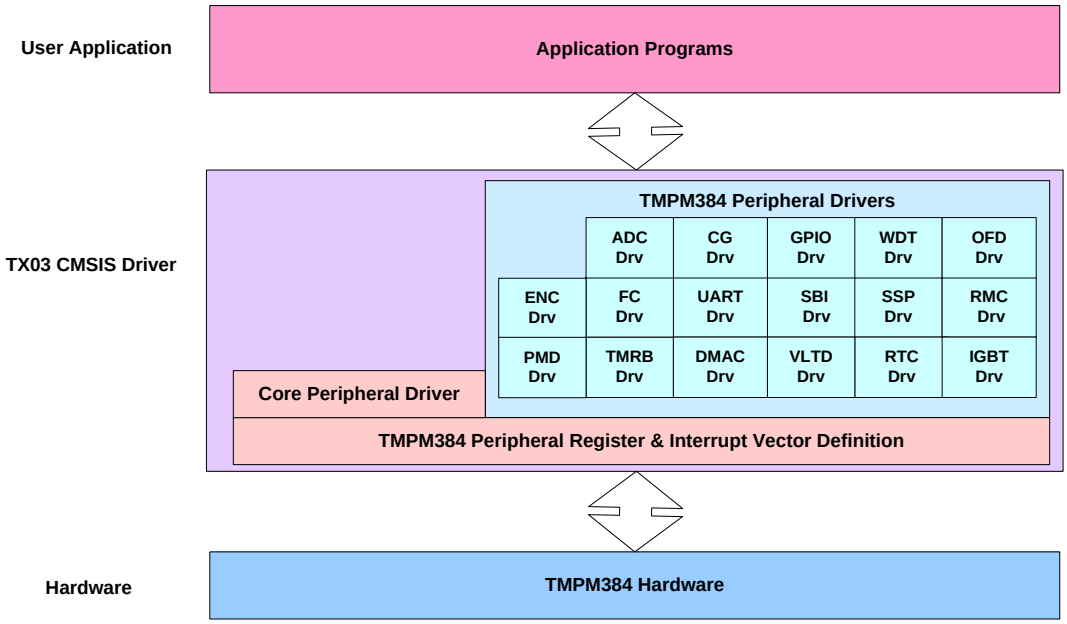
Index

1 General description

The example programs described hereafter are specially designed for TOSHIBA TMPM384 MCU. The programs can be executed individually each to show the main MCU functions. Users can utilize some portions of the programs and integrate them into users' own programs to perform customized functions.

2 Overview

User application utilizes TX03 peripherial driver as following.



3 Build-in hardware usage

Hardware	Channel	Use presence, use
CG	Clock Gear	Used for CG demo
	PLL	PLL on
Standby mode	-	Use SLEEP mode
SysTick	-	Unused
DMAC	channel 1	Used for DMAC demo
Watch dog timer (WDT)	-	Used for WDT demo
External interrupt (INT)	INT0	Used for CG demo
SIO	SIO1	Used for UART & DMAC demo Tx
Synchronous serial port (SSP)	SSP0	Used for SSP transfer demo Tx
	SSP1	Used for SSP transfer demo Rx

Hardware	Channel	Use presence, use
Serial bus (SBI)	SBI0	Used for Master Tx
	SBI1	Used for Slave Rx
Remote control signal processor (RMC)	RMC0	Used for RMC signal receive
16-bit timer	TMRB0	Used for TMRB: General Timer
	TMRB6	Used for TMRB: PPG Output
16-bit multi-purpose timer	MPT0	Used for IGBT demo: PPG output
	MPT1	Used for PMD demo: Phase Output
RTC	-	Used for RTC demo
OFD	-	Used for OFD demo
VLTD	-	Used for low voltage detection
Encoder input function	ENC0	Used for ENC Roller demo
12/10-bit A/D converter	AIN8	Used for ADC data read
Flash	-	Used for FC demo

4 Pin Usage

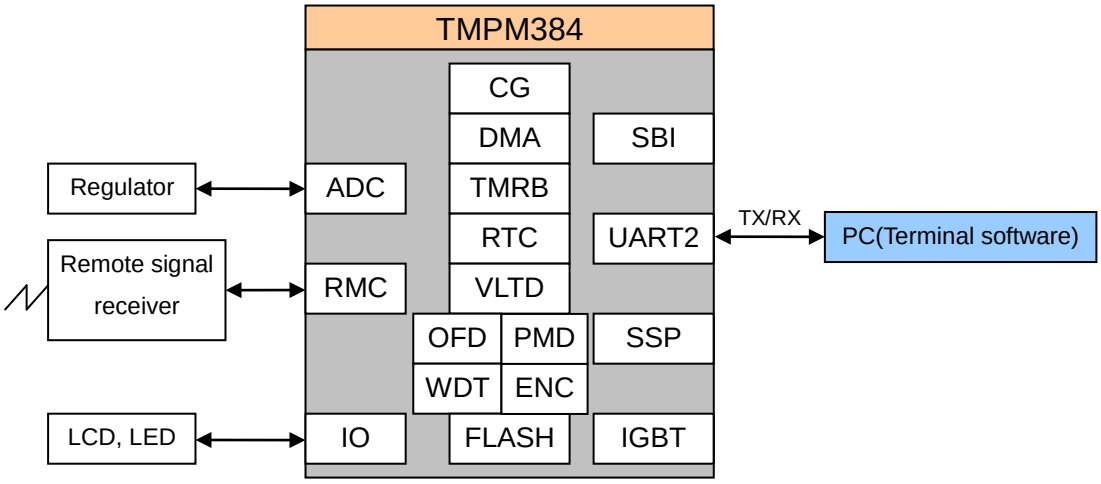
The example programs are tested on TOSHIBA Evaluation Board for TMPM384FDFG.
Following is pin usage for example programs on TOSHIBA Evaluation Board for TMPM384FDFG.

Pin No.	Name	Usage
92	PH0/AIN0/INT0	SW0/4
93	PH1/AIN1/INT1	SW1/5
94	PH2/AIN2	SW2/6
95	PH3/AIN3	SW3/7
33	PT0/TB8IN	LED0
34	PT1/TB8OUT	LED1
35	PT2/TB9IN	LED2
36	PT3/TB9OUT	LED3
20	PC1/XO0/SP0DI/SCL0/SI0	Master (SBI0) SCL0
21	PC0/UO0/SP0DO/SDA0/SO0	Master (SBI0) SDA0
29	PG0/UO1/SDA1/SO1	Slave (SBI1) SDA1
30	PG1/XO1/SCL1/SI1	Slave (SBI1) SCL1
14	PC7, MT0IN	IGBT0 Trigger input
15	PC6, GEMG0	IGBT0 EMG input
17	PC4, MTOUT00	IGBT0 PPG output 0
16	PC5, MT0OUT10	IGBT0 PPG output 1
52	PA5,TB6OUT	TMRB6 PPG output
50	PA3,RXIN	RMC Recieve input
18	PC3	SSP SP0FSS
19	PC2	SSP SP0CLK
20	PC1	SSP SP0DI
21	PC0	SSP SP0DO
52	PA5	UART1 TX
65	PN0	SSP SP1DO
66	PN1	SSP SP1DI
67	PN2	SSP SP1CLK
81	PN3	SSP SP1FSS
100	PI0/AIN8	ADC AIN8
12	PD0/ENCA0/TB5IN/INTC	ENCA0
11	PD1/ENCB0/TB5OUT	ENCB0
116	DVSS	GND pin
117	DVDD5	+5V
29	PG0 / UO1 / SDA1/SO1	PMD UO1 output

Pin No.	Name	Usage
30	PG1 / XO1 / SCL1/SI1	PMD XO1 output
31	PG2 / VO1 / SCK1	PMD VO1 output
32	PG3 / YO1	PMD YO1 output
43	PG4 / WO1 / MTOUT01 / MTTB1OUT	PMD WO1 output
44	PG5 / ZO1 / MTOUT11 MTTB1IN	PMD ZO1 output
45	PG6 / EMG1 / GEMG1	PMD EMG1 input

5 Development Environment

Following is development environment:



1. Hardware board:
   TOSHIBA Evaluation Board for TMPM384FDFG

2. Development tool:
   - IAR:
     1) J-Link: IAR J-Link-7.0/8.0
     2) IDE: IAR Embed Workbench for ARM version 6.40.2
   - KEIL:
     1) u-Link: Realview ULINK2
     2) IDE: KEIL uVision MDK version 4.54

***Note:** For ENC and PMD modules, the development tool version is different

- IAR:
  1) J-Link: IAR J-Link-ARM 7.0 / J-Link 6.0
  2) IDE: IAR Embedded workbench 6.50.1 version
- KEIL:
  1) IDE: KEIL uVision 4.60

6 Functions

6-1 Operation mode

There are five operation modes for TMPM384: NORMAL, SLOW, IDLE, SLEEP and STOP mode.
IDLE, SLEEP and STOP mode are low power mode.
To shift to lower power mode, in system control register STBYCR0<STBY2:0>, select the IDLE, SLEEP or STOP mode, and run the WFI (Wait For Interrupt) command.

- **NORMAL mode:**

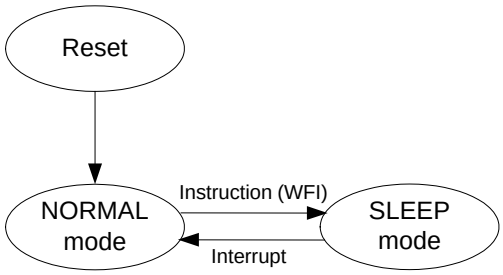
This mode is to operate the CPU core and the peripheral hardware by using the high-speed clock(fosc1 or fosc2). It is shifted to the NORMAL mode with fosc2(internal high-speed oscillator) after reset.

Note: Only SLEEP mode is demonstrated in driver example.

- **SLEEP mode:**

The internal low-speed oscillator (fs), real time clock and RMC can operate. By releasing the SLEEP mode, the device returns to the preceding mode of the SLEEP mode and starts operation.
RTC interrupt, RMC interrupt and the extern interrupt can release the SLEEP mode.

Note: The sample program of SLEEP mode is integrated into CG example.



6-2 ADC

Change the value of *VR2*, which is connected to *AIN8*(pin100). The voltage on it will be measured and used to change the loop blinking speed of the 4 LEDs in board.
The higher the voltage is, the faster the loop blinking speed.

The driver example step:

1. Set ADC clock prescale and sample hold time.
2. Select trigger and AD channel
3. Disable low power mode.
4. Enable ADC module.
5. Start ADC.
6. Check if ADC is finished, get the result and use it to adjust the LED loop blinking speed.
7. Repeat step 6 above.

6-3 CG

6-3-1 Power mode change

Change the CPU operation mode. Only 2 modes are supported, NORMAL and SLEEP. Toggle switch to switch the power mode.

Current mode	Action (Switch)	Operation	LED display
NORMAL	Turn on SW1	NORMAL→ STOP1	All LED turns off.
STOP1	Turn off SW1，turn off SW0((low active)).	STOP1 →NORMAL	All LED turns on.

6-4 DMAC

6-4-1 Memory to peripheral

This is a simple application based on the TMPM384 Peripheral Driver (DMAC), which transfer data from memory to memory by DMA channel 1. DST_Buffer[] will receive the data which transmitted by DMA, and compare result(PASSED or FAILED) will be saved in the variable of TransferResult.

6-5 FLASH

This example demonstrates the feature of flash APIs such as erase and write operation.
The demo runs in Normal mode and User Boot Mode of Single chip mode.
－Reset procedure: includes Mode judgment, Programming routine(flash APIs), Copy routine
・Mode judgment routine: judge to enter User Boot Mode or Normal Mode
・Copy routine: copy programming routine(flash APIs) from flash to RAM
・Programming routine: runs at RAM and swap Program A and Program B code in flash

－Program A/B (Reset procedure) are programmed in flash block in advance.

－Program A/B(A: LED0 blinks and LCD displays ”DEMO A”,

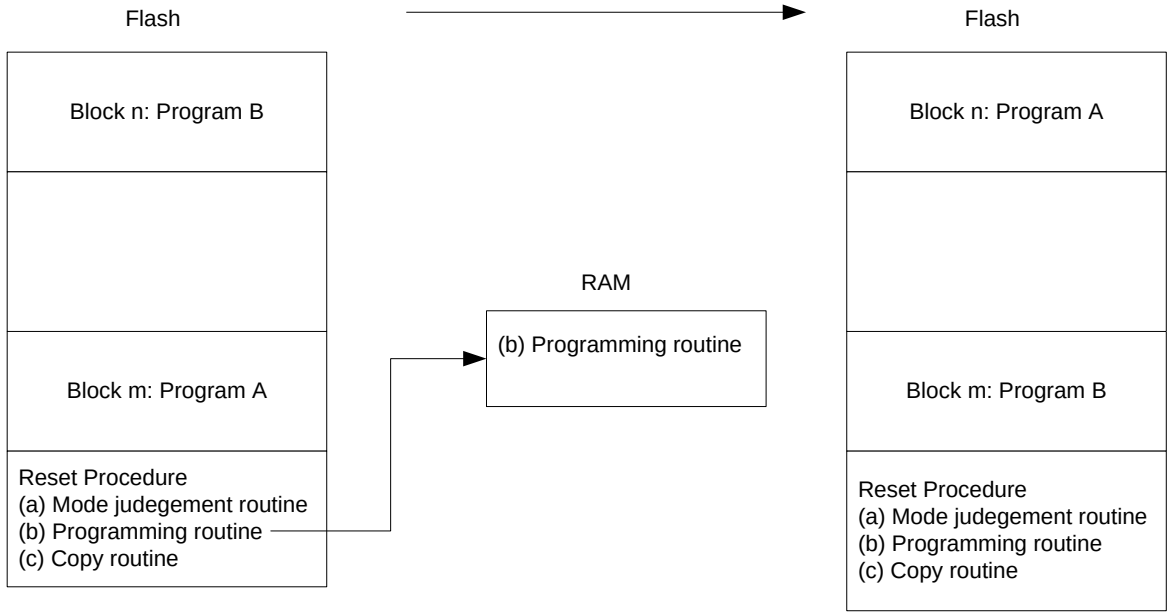
B: LED1 blinks and LCD displays “DEMO B”)

By default, the program A will run firstly

－Toggle switch SW0 is used for mode judgment in Reset procedure

SW0 turned on → User boot mode

SW0 turned off → Normal mode



Demo sequence

(1) Power on

The demo board runs Reset Procedure.

Then initial program A stored in flash runs.

LED0 blinks, and LCD displays “DEMO A”.

(2) Press RESET button while SW0 is turned on, and release SW0 until LCD displays “User Boot Mode”.

The demo board runs Reset Procedure: Programming routine is copied to RAM by Copy routine.

Then run Programming routine to swap program A and B in flash.

During this period, LCD displays information such as “RAM transferring ……”, “FLASH swapping ……” and “Finished Restart ……”.

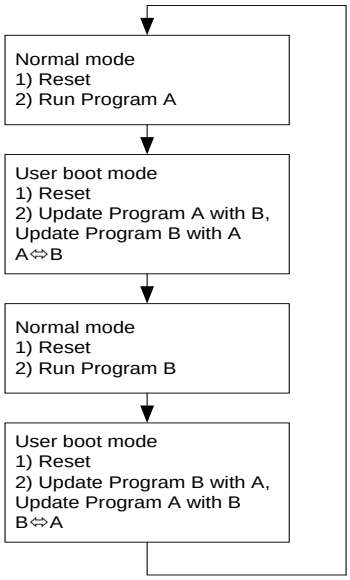
(3) After LCD has displayed “Finished Restart ……”, the demo will reset by software automatically.

Then program B stored in flash runs.

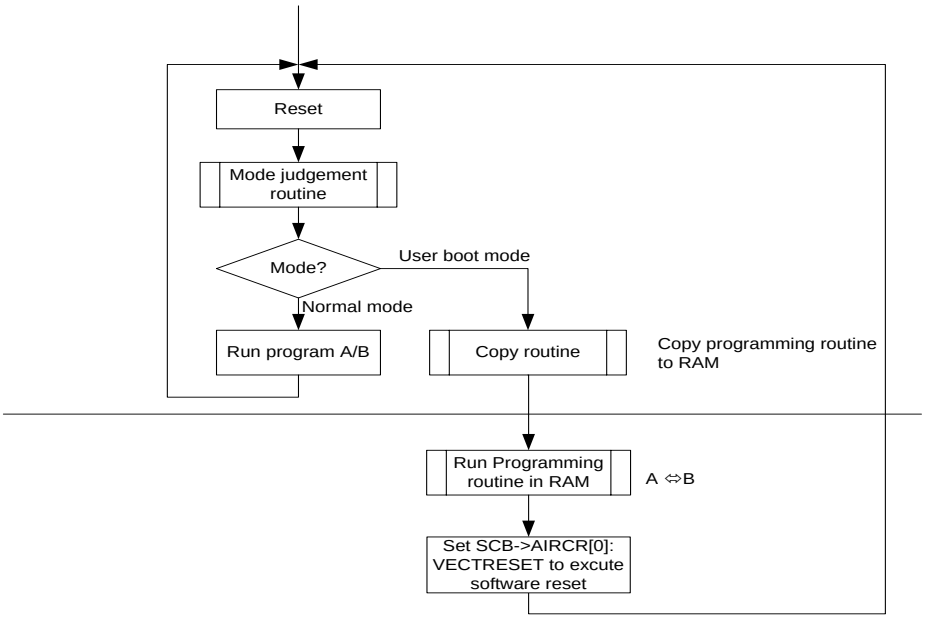
LED1 blinks, and LCD displays “DEMO B”.

(4) Press RESET button again while SW0 is turned on, and release SW0 until LCD display information “User Boot Mode”.
Same as step (2).

(5) After LCD has displayed “Finished Restart ……”, the demo will reset by software automatically.
Then program A stored in flash runs.
LED0 blinks, and LCD displays “DEMO B”.



・**Demo process flow**



When the demo entering user boot mode, LCD will display as below:

User Boot Mode

(Reset Procedure)

While RAM transferring or flash programming, LCD displays the current processing.

LCD display sample:

RAM transferring ……	FLASH swapping ……	Finished Restart ……

(Copy routine) (Programming routine) (Programming routine)

6-6 GPIO

This is a simple application based on the Peripheral Driver (GPIO), use GPIO API functions to configure LED and switch, turn on the LED, or turn off the LED.

6-7 OFD

This is a simple application based on the Peripheral Driver (OFD).
When the clock exceeds the OFD detection frequency range, OFD will generate a reset for I/O. Then software will be reset. The OFD reset flag will be set. If the flag is detected, OFD will be disabled and LED will be turned on.

6-8 IGBT

6-8-1 Single PPG Output

This example program demonstrates how to use external trigger to control the PPG output by TMPM384 IGBT driver and EMG function. The MPT channel 0 (IGBT0) is used and the PPG wave is generated from MTOUT00.

Pin assignment:

Function	Pin Name	Pin No.
Trigger input	MT0IN / PC7	14
PPG output	MTOUT00 / PC4	17
EMG input	GEMG0 / PC6	15

Demo procedure:

1. Use a wire to connect port GEMG0 with DVDD to pull it high and keep it to high.
2. Connect MTOUT00 to oscilloscope.
3. Power on, LED0 on and no waveform is observed from MTOUT00.
4. Touch MT0IN with GND to produce a falling edge.
5. The IGBT timer starts to run because of the falling edge and at the same time PPG waveform can be observed from MTOUT00. PPG cycle is 50us with duty 50%.
6. Connect GEMG0 with GND, then PPG output stops immediately and LED1 lights on.
7. Connect GEMG0 with DVDD again, and EMG state cancels with LED1 off, then repeat step 4 to observe the waveform.

6-8-2 Double PPG Output

This example program demonstrates how to use command start to control two PPG output by TMPM384 IGBT driver and EMG function. The MPT channel 0 (IGBT0) is used and two PPG waves are generated from MTOUT00 and MTOUT10.

Pin assignment:

Function	Pin Name	Pin No.
PPG output 0	MTOUT00 / PC4	17
PPG output 1	MTOUT10 / PC5	16
EMG input	GEMG0 / PC6	15

Demo procedure:

1. Use a wire to connect port GEMG0 with DVDD to pull it high.
2. Connect MTOUT00 and MTOUT10 to oscilloscope.
3. Switch SW7 to OFF.
4. Power on, LED0 on and no waveforms are observed from MTOUT00 or MTOUT10.
5. Switch SW7 to ON.
6. The IGBT timer starts to run and at the same time PPG waveform can be observed from MTOUT00 and MTOUT10. Both PPG waveforms' cycle is 50us with duty 40 %(high level: 20us) and MTOUT10 delays MTOUT00 with 25us.
7. If switch SW7 to OFF, PPG waveform disappears.
8. Connect GEMG0 with GND, then PPG output stops immediately and LED1 lights on.

9. Connect GEMG0 with DVDD again, and EMG state cancels with LED1 off, then repeat step 5 to observe the waveform.

6-9 RMC

6-9-1 RMC receiving

This module intends to catch the remote signal data and decode it. Decoded data will be displayed on terminal on IAR. Custom code or address code and command code will be displayed in Hex format.

Display format: TMPM384 RMC Demo
RMC_1: XX XX
RMC_1: YY YY

Format	Terminal soft display
1	RMC_1：
2	RMC_2：
3	RMC_3：
4	RMC_4：
5	RMC_5：

6-10 RTC

Use the internal RTC to display the time and date on LCD.

Display the time and date on LCD and update the date and time.

Initial setting: **2010/10/22 12:50:55**, 24hours format.

Update interval: 1 second.

LCD displays:

2010/10/22 12:50:55

6-11 SBI

This function intends to support the I2C bus slave mode.

Connect two I2C buses (SBI0 & SBI1) on TOSHIBA TMPM384 evaluation board.
One is working as I2C master, and the other is working as I2C slave.
Use SBI interrupt to handle I2C bus read/write.
Address of I2C slave: 0xB0.
I2C Slave (SBI1) receives “TOSHIBA” from I2C Master (SBI0).

Received results can be checked by RAM variable gI2CrxData[] in debugger.

Note: In order to run this sample software, use TOSHIBA TMPM384 evaluation board which must be modified.
Connect the pin PC1(SCL0) and pin PG1(SCL1).
Connect the pin PC0(SDA0) and pin PG0(SDA1).

6-12 SIO/UART

6-12-1 Transmission from UART0 To UART2

In the SIO sample program, UART0 is used to send data and UART2 is used to receive data.
String “TMPM384” is sent from UART0.
UART2 receives the data, and data will get in RxBuffer. Set one breakpoint before the function ResetIdx(),when program stop in the breakpoint you can read the RxBuffer = "TMPM384".
UART Settings:
Communication Channel: UART0 and UART2
Data length : 8 bits
Parity bit : None
Stop bit : 1 bit
Baud rate : 115200bps
Flow control : None

6-12-2 Retarget

This function intends to retarget the stdin and stdout of the C lib.
Stdin and stdout are retargeted to UART1. Then application code can use printf() and getchar() to output to or input from serial port.

6-12-3 UART FIFO

In this sample program, UART0 sent the data "TMPM3841" to UART1 use FIFO, and at same time UART0 receive the data "TMPM3842" which send from UART1 and also use FIFO.
Set one breakpoint before the function ResetIdx(),when program stop in the breakpoint you

can read the RxBuffer = "TMPM3842",and RxBuffer1 = "TMPM3841".

6-12-4 SIO

This demo will show synchronously transfer and receive usage of SIO module in TMPM384
It use the channel SIO0, SIO1 and transfer data synchronously between them.(connect TXD0 with RXD1, connect TXD1 with RXD0, connect sclk0 with sclk1)

6-13 SSP

6-13-1 SSP0 to SSP1 via DMAC

For M384, there are 2 SSP channels, SSP0 is set as SPI Mast while SSP1 is set as SPI Slave, then connect correspond pins, use DMA to transmit/receive infinite self increased data.

Note: TOSHIBA TMPM384 evaluation board must be modified as below:

	SSP0 pins	Note	SSP1 pins
1	PC3/SP0FSS , pin18	Connect PC3 to PN3 (pin18 to 81)	PN3/SP1FSS , pin81
2	PC2/SP0CLK, pin19	Connect PC2 to PN2 (pin19 to 67)	PN2/SP1CLK , pin67
3	PC1/SP0DI, pin20	Connect PC1 to PN0 (pin20 to 65)	PN1/SP1DI, pin66
4	PC0/SP0DO, pin21	Connect PC0 to PN1 (pin21 to 66)	PN0/SP1DO, pin65

* SSP0 to SSP1 via DMAC demo should run on TOSHIBA TMPM384 Evaluation Board.

6-13-2 SSP0 Self Loop Back

Infinite data is sent and checked if the received data is OK or not. And if transmission is set to lowest speed mode(lowest bit rate), what happens will be checked.

6-14 TMRB

6-14-1 General Timer

This function implements a general timer utilizing MCU timer.
Timer trailing timing is set to 1ms.

Use this timer for LED blinking and the period is 1s (500ms ON and 500ms OFF).

.

6-14-2 PPG Output

Use Toggle Switch 1 to change PPG waveform. Connect PA5/TB6OUT to oscilloscope.
LeadingTiming can be changed as following:
10%, 25%, 50%, 75%, 90%
Power on to enter PPG mode and starts the PPG output.
Change the leading timing every switch changing:
10% ->25%--> 50%-->75% --> 90%-->10%

6-15 VLTD

This application implements power supply voltage status detecting by VLTD.
When the power supply voltage is lower than the detection voltage, LED3 is turned on and a “Low voltage!” message is printed (In DEBUG mode). When the power supply voltage is normal again, LED3 is turned off and a “High voltage!” message is printed (In DEBUG mode).

6-16 WDT

The watchdog timer cannot be used in the STOP mode where high-speed frequency clock is stopped.
Watch dog timer is enabled after reset, and is disabled in SystemInit().

The driver example step:

1. Initialize WDT by setting detection time and selecting WDT interrupt as counter overflow operation.
2. There are two demos for WDT in which DEMO2 is switched by macro definition.

DEMO1:

When timer is overflow, NMI interrupt is generated and then WDT will be cleared.

DEMO2:

WDT clear code will be written to the watchdog timer control register, and watch dog timer counter will be cleared.

6-17 ENC

This example detects the rolling of mouse wheel by TMPM384 ENC driver.

Demo procedure:

1. Use a wire to connect the pinA (see diagram 1) of mouse wheel encoder with the pin12 (ENCA0) of TMPM384FDFG MCU board.
2. Use a wire to connect the pinB (see diagram 1) of mouse wheel encoder with the pin11 (ENCB0) of TMPM384FDFG MCU board.
3. Connect the USB of mouse with the PC, the Com (see diagram 1) of mouse wheel encoder will connect with 5V.
4. After running the program, LED1 will blink.
5. While rolling the mouse wheel forward, LED1 will blink faster and faster. When the blink speed reaches to the maximum, it will back to the minimum speed.
6. While rolling the mouse wheel backward, LED1 will blink slower and slower. When the blink speed reaches to the minimum, it will back to the maximum speed.

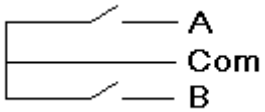


Diagram 1

6-18 PMD

This example demonstrates the phase outputs in PMD module.

Demo procedure:

1. Open the project and choose one demo to define: DEMO_U_PHASE or DEMO_3_PHASE. And then run the demo.
2. Use a wire to connect the EMG1 (pin 45) with the DVSS (pin 116) to pull it low. (LED1 will light on) It means that PMD is in EMG protection.
3. Use a wire to connect the EMG1 (pin 45) with the DVDD5 (pin 117) to pull it high. (LED1 will light off) It means that PMD is in normal operation.
4. Connect UO1 (pin 29) to oscilloscope to observe the waveform.
5. Connect XO1 (pin 30) to oscilloscope to observe the waveform.
6. Connect VO1 (pin 31) to oscilloscope to observe the waveform.
7. ConnectYO1 (pin 32) to oscilloscope to observe the waveform.
8. Connect WO1 (pin 43) to oscilloscope to observe the waveform.
9. Connect ZO1 (pin 44) to oscilloscope to observe the waveform.

Phenomenon:

If the DEMO_U_PHASE is defined, the phase outputs are same.

The waveform duty cycle of UO1 is 25%, the upper output is 5V, and the period is 820us.XO1 is the inverting of UO1.

VO1 and WO1 are same with UO1, and YO1 and ZO1 are same with XO1

If the DEMO_3_PHASE is defined, the phase outputs are different.

The waveform duty cycle of UO1 is 25%, the upper output is 5V, and the period is 820us. XO1 is the inverting of UO1.
The waveform duty cycle of VO1 is 50%, the upper output is 5V, and the period is 820us. YO1 is the inverting of VO1.
The waveform duty cycle of WO1 is 75%, the upper output is 5V, and the period is 820us. ZO1 is the inverting of WO1.

7 Software

This software project creates the sample applications based on TOSHIBA Evaluation Board for TMPM384FDFG (TX03-BASEG-A + M384FDFG-XFRM-G-A + M384FDFG).

Use current driver software and IAR EWARM or KEIL MDK to implement the sample applications. Create an independent project for each feature.

The workspace structure and project names are defined as following:

IAR EWARM:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm384_wdt_int.c
        ├─TX03_CMSIS
        │   │   system_TMPM384.c
        │   │   system_TMPM384.h
        │   │   TMPM384.h
        │   │
        │   └─startup
        │         startup_TMPM384.s
        │
        ├─TX03_Periph_Driver
        │   ├─inc
        │   │   tmpm384_wdt.h
        │   │   tmpm384_gpio.h
        │   │   tx03_common.h
        │   │
        │   └─src
        │         tmpm384_wdt.c
        │         tmpm384_gpio.c
        └─TMPM384-EVAL
                  led.c
                  led.h
```

KEIL MDK:

```
├─WDT_NMI
        ├─APP
        │   main.c
        │   tmpm384_wdt_int.c
        │
        ├─TX03_CMSIS
        │   │   system_TMPM384.c
        │   │   system_TMPM384.h
        │   │   TMPM384.h
        │   │
        │   └─startup
        │         startup_TMPM384.s
        │
        ├─TX03_Periph_Driver
        │    tmpm384_wdt.c
        │    tmpm384_gpio.c
        │
        └─TMPM384-EVAL
             led.c
```

7-1 ADC

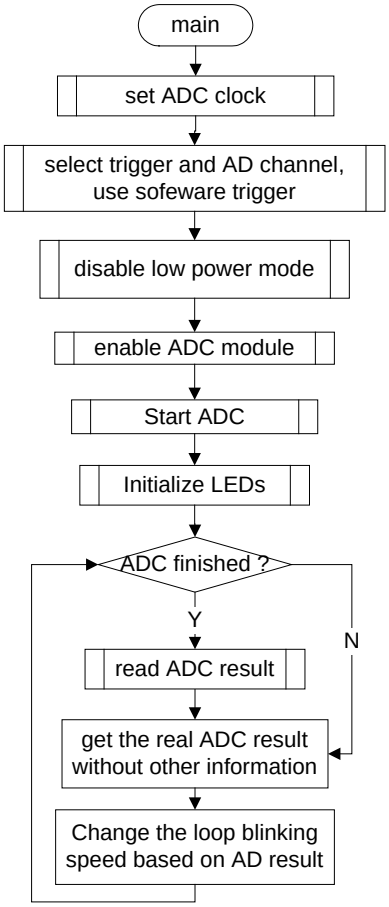
7-1-1 Example: Read AD Value By Software Trigger

This is a simple example based on the TX03 Peripheral Driver (ADC, GPIO).

The example includes:

1. ADC configuration and initialization
2. Start AD conversion and read AD conversion result
3. Use AD conversion result to adjust the loop blinking speed.

- **Flowchart:**



- **Code and Explanation for the Example**

At first, set ADC clock, select channel and enable the software trigger mode, set Low Power mode.

```
/* 1. set ADC clock */
    ADC_SetClk(ADC_HOLD_FIX, ADC_FC_DIVIDE_LEVEL_NONE);

/* 2. select trigger and AD channel, this time we use sofeware trigger, */
/*      remember to input with macro TRG_ENABLE() */
ADC_SetSWTrg(ADC_REG0, TRG_ENABLE(ADC_CHANNEL));

/* 3. to let ADC module work, we must disable low power mode */
ADC_SetLowPowerMode(DISABLE);
```

Enable ADC module and start ADC.

```
/* 4. enable ADC module */
ADC_Enable();

/* 5. now start ADC */
ADC_Start(ADC_TRG_SW);
```

After started AD conversion, wait the flag for ADC conversion finished, then read ADC conversion result and use it to adjust the LEDs loop blinking speed.

```
/* check ADC module state */
```

```
adcState = ADC_GetConvertState(ADC_TRG_SW);

if (adcState == DONE) {
      /* read ADC result when it is finished */
      adResult = ADC_GetConvertResult(ADC_REG0);

      /* Get the real ADC result without other information */
      /* "/256" is to limit the range of AD value */
      timeUp = 16U - adResult.Bit.ADResult / 256U;
/* use 'timeUp' above to adjust the loop blinking speed */

/* software trigger, need to trig it again */
      ADC_Start(ADC_TRG_SW);
} else {
      /* Do nothing */
}
```

7-2 CG

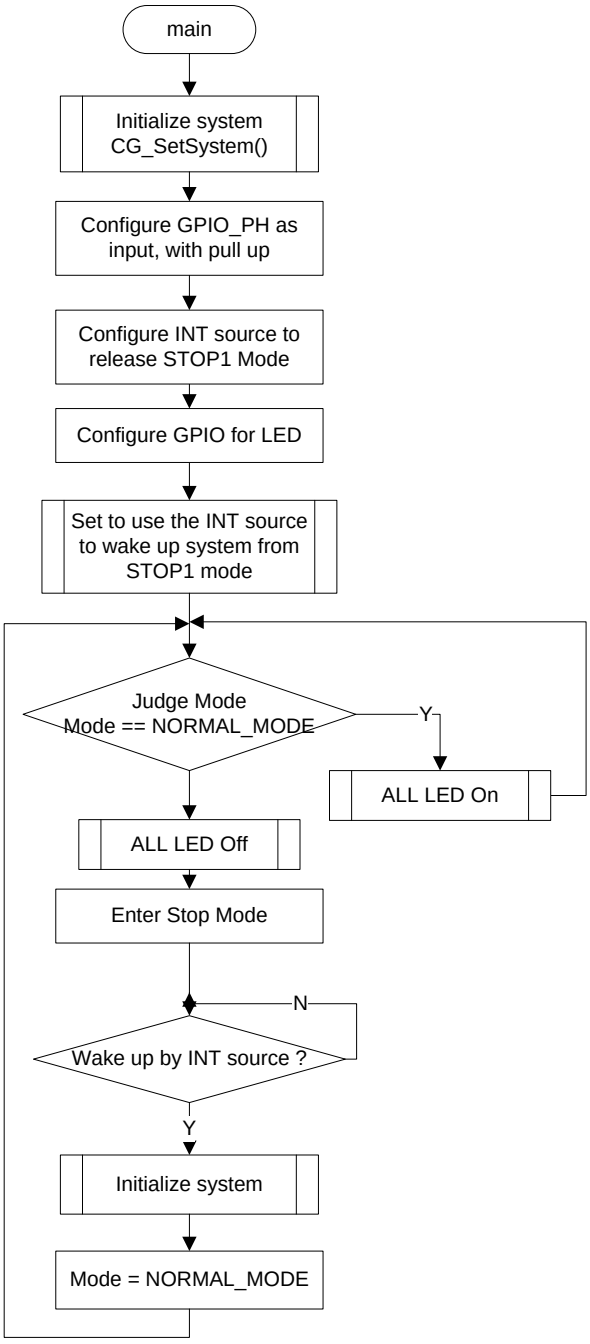
7-2-1 Example: Power mode change

This is a simple example based on the TX03 Peripheral Driver (CG and GPIO).

The example includes:

1. Basic setup operation of CG
2. How to switch between normal mode and sleep mode
3. How to enable multiple clock circuit

- **Flowchart**



- **Code and Explanation for the Example**

The following simple example is based on TX03 Peripheral Driver (CG), which will switch between normal mode and sleep mode.

```
void CG_SetSystem(void)
{
    /* Set fgear = fc/2 */
    CG_SetFgearLevel(CG_DIVIDE_2);
```

```
    /* Set fperiph to fgear */
    CG_SetPhiT0Src(CG_PHIT0_SRC_FGEAR);
    CG_SetPhiT0Level(CG_DIVIDE_64);
    /* select external oscillator */
    CG_SetFoscSrc(CG_FOSC_OSC1);
    CG_SetPortM(CG_PORTM_AS_HOSC);
    /* Enable high-speed oscillator */
    CG_SetFosc(CG_FOSC_OSC1,ENABLE);
    /* Set low power consumption mode stop */
    CG_SetSTBYMode(CG_STBY_MODE_STOP);
    /* Set pin status in stop mode to "active" */
    CG_SetPinStateInStopMode(ENABLE);
    /* Set up pll and wait for pll to warm up, set fc source to fpll */
    CG_EnableClkMulCircuit();
}
```

Configure Keys, LED pins, INT source pins

Configure the I/O pins for Key, LED and INT source

```
        /* Initialize system */
        CG_SetSystem();

        /* Configure GPIO */
        GPIO_SetInput(GPIO_PH, GPIO_BIT_0);      /* Set port H pin0 to input */
        GPIO_EnableFuncReg(GPIO_PH, GPIO_FUNC_REG_1, GPIO_BIT_0);
        /* Set port H pin0 for INT0 */
        GPIO_SetInput(GPIO_PH, GPIO_BIT_1);      /* Set port H pin1 to input */
        /* Configure GPIO to LED */
        LED_Init();
```

Configure external interrupt to wake up system

Confiure external interrupt INT0 to wake up system. Clear interrupt pending request, then enable INT0.

```
        /* Disable interrupts */
    __disable_irq();
    CG_ClearINTReq(CG_INT_SRC_0);
    /* Set external interrupt to wake up system */
    CG_SetSTBYReleaseINTSrc(CG_INT_SRC_0,
                                    CG_INT_ACTIVE_STATE_FALLING, ENABLE);
    NVIC_ClearPendingIRQ(INT0_IRQn);
    NVIC_EnableIRQ(INT0_IRQn);
    __enable_irq();
```

Setup to enter STOP mode

Prepare for entering stop mode. Set warm up time, use __WFI () instruction to enter stop mode.

```
/* CG_NormalToStop */
void CG_NormalToStop(void)
{
    /* Set CG module: Normal ->Stop mode */
    CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC1,CG_WUODR_EXT);
    /* Enter stop mode */
    __WFI();
}
```

Enable multiple clock circuit

First set PLL value, and then set warm up time and wait warm up time is completed. Finally set fPLL as fc source.

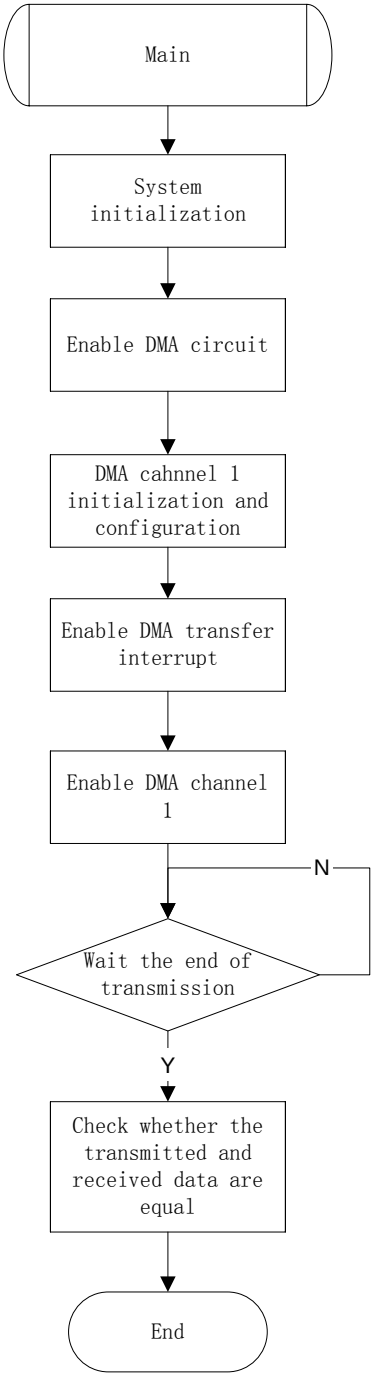
```
Result CG_EnableClkMulCircuit(void)
{
    Result retval = ERROR;
    WorkState st = BUSY;
    retval = CG_SetPLL(ENABLE);
    if (retval == SUCCESS) {
        /* Set warm up time */
        CG_SetWarmUpTime(CG_WARM_UP_SRC_OSC1,CG_WUODR_PLL);
        CG_StartWarmUp();

        do {
            st = CG_GetWarmUpState();
        } while (st != DONE);

        retval = CG_SetFcSrc(CG_FC_SRC_FPLL);
    } else {
        /*Do nothing */
    }

    return retval;
}
```

7-3 DMAC

7-3-1 Example: Memory to peripheral

This is a simple application based on the TMPM384 Peripheral Driver (DMAC), which transfer data from memory to memory by DMA channel 1.

- **Flowchart**



- **Code and Explanation for the Example**

At first, create a DMAC_InitTypeDef structure and fill all the data fields. For example,

```
DMAC_InitTypeDef DMAC_InitStruct;

DMAC_InitStruct.TxDirection = DMAC_MEMORY_TO_MEMORY;
DMAC_InitStruct.SrcAddr = (uint32_t) SRC_Buffer;
```

```
DMAC_InitStruct.DstAddr = (uint32_t) DST_Buffer;
DMAC_InitStruct.SrcIncrementState = ENABLE;
DMAC_InitStruct.DstIncrementState = ENABLE;
DMAC_InitStruct.SrcBitWidth = DMAC_WORD;
DMAC_InitStruct.DstBitWidth = DMAC_WORD;
DMAC_InitStruct.SrcBurstSize = DMAC_4_BEATS;
DMAC_InitStruct.DstBurstSize = DMAC_4_BEATS;
DMAC_InitStruct.TxSize = BUFFER_SIZE;
DMAC_InitStruct.TxINT = ENABLE;
```

Then enable DMA circuit, initialize and configure DMA channel. Then enable DMA transfer interrupt and DMA channel.

```
DMAC_Enable();
DMAC_Init(myChannel, &DMAC_InitStruct);
DMAC_TxINTConfig(myChannel, DMAC_INT_TX_END, ENABLE);
DMAC_SetDMAChannel(myChannel, ENABLE);
```

After the setting above, wait for the end of transmission.

```
while (TxEndFlag != DONE) {
        /* Do nothing */
}
```

The flag of transfer end will be set in ISR of DMAC transfer end interrupt.

```
state = DMAC_GetINTReq();

if (state.Bit.CH1_INTReq == 1U) {
    tmp = DMAC_GetTxINTReq(DMAC_CHANNEL_1);
    if (tmp == DMAC_TX_END_REQ) {
        TxEndFlag = DMAC_GetChannelTxState(DMAC_CHANNEL_1);
        DMAC_ClearTxINTReq(DMAC_CHANNEL_1, DMAC_INT_TX_END);
    } else {
        /* Do nothing */
    }
} else {
    /* Do nothing */
}
```

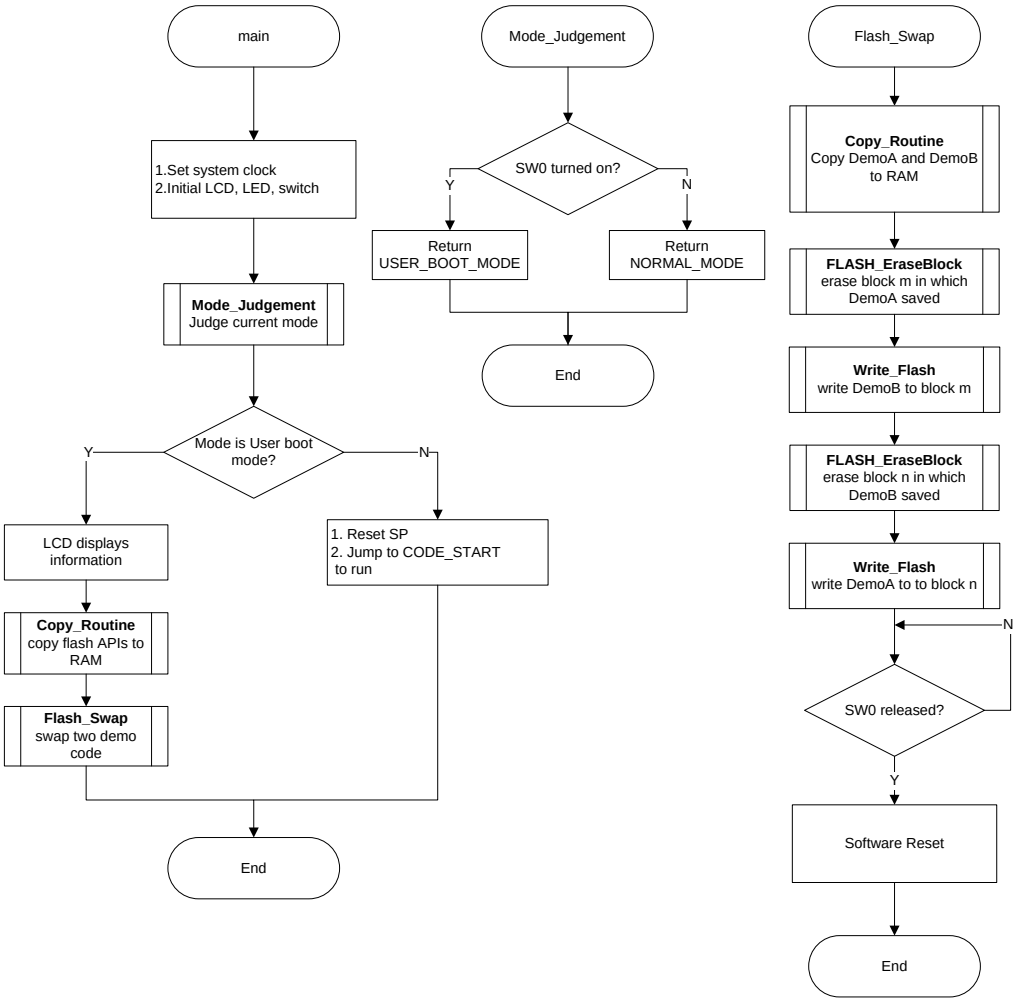
7-4 FLASH

This is a simple example based on the TX03 Peripheral Driver (FLASH, GPIO).

The example includes:

1. On-board programming method of Flash Memory (Rewrite/Erase)
2. Operation mode: Single chip mode (Normal mode and User boot mode)
3. Use User boot mode to update code in Flash Memory

- **Flowchart**



- **Code and Explanation for the Example**

At first initialize LED LCD and switch. SW0 (GPIO) is used to judge current mode when reset and LCD will display current running routine and Flash operation information.

```
    LED_Init();
    SW_Init();
    LCD_Configuration();
```

Use function Mode_Judgement() to judge which mode will be entered after reset.

```
uint8_t Mode_Judgement(void)
{
    return (SW_Get(SW0) == 1U) ? USER_BOOT_MODE : NORMAL_MODE;
}
```

If the SW0 is not turned on when reset, the routine will enter normal mode. Then the routine will reset SP and jump to address “CODE_START” to run. Demo A saved in at address “CODE_START” which belongs to block m, so Demo A will run (LED0 blinks).

```
#if defined ( __CC_ARM     )          /* RealView Compiler */
        ResetSP();                    /* reset SP */
```

```
#elif defined ( __ICCARM__ )       /* IAR Compiler */
            asm("MOV R0, #0");          /* reset SP */
            asm("LDR SP, [r0]");
#endif
            SCB->VTOR = DEMO_START_ADDR;       /* redirect vector table */
            startup = CODE_START;
            startup();                  /* jump to code start address to run */
```

If the SW0 is turned on when reset, the routine will enter user boot mode. LCD will display information to user. Then the routine will copy APIs for flash operation from address “FLASH_API_ROM” in Flash memory to address “FLASH_API_RAM” in RAM, because Flash memory cannot erase/program itself when runs the code at the same time.

```
            LCD_Display("User Boot Mode", "");          /* enter user boot mode */
            LCD_Display("RAM transferring", "......");

            Copy_Routine(FLASH_API_RAM, FLASH_API_ROM, SIZE_FLASH_API);
          /* copy flash API to RAM */

            LCD_Display("Flash swapping", "......");
```

After copying Flash operation APIs to RAM, the routine will jump to function Flash_Swap(), this function has been copied from Flash memory to RAM by using Copy_Routine() above.

In function Flash_Swap(), firstly it will copy Demo A and B from Flash memory to RAM, then call function FLASH_EraseBlock() and Write_Flash() to erase and program Flash memory. The code of Demo A and B will be exchanged in Flash memory.

```
    Copy_Routine(DEMO_A_RAM,  DEMO_A_FLASH,  SIZE_DEMO_A);              /*
copy A to RAM */
    Copy_Routine(DEMO_B_RAM,  DEMO_B_FLASH,  SIZE_DEMO_B);              /*
copy B to RAM */

    if (FC_SUCCESS == FC_EraseBlock((uint32_t) DEMO_A_FLASH)) { /* erase A */
        /* Do nothing */
    } else {
        return ERROR;
    }

    if    (FC_SUCCESS    ==    Write_Flash(DEMO_A_FLASH,    DEMO_B_RAM,
SIZE_DEMO_B)) {       /* write B to A */
        /* Do nothing */
    } else {
        return ERROR;
    }

    if (FC_SUCCESS == FC_EraseBlock((uint32_t) DEMO_B_FLASH)) { /* erase B */
        /* Do nothing */
    } else {
        return ERROR;
    }

    if    (FC_SUCCESS    ==    Write_Flash(DEMO_B_FLASH,    DEMO_A_RAM,
SIZE_DEMO_A)) {       /* write A to B */
```

```
        /* Do nothing */
    } else {
        return ERROR;
    }
```

LCD displays the information to indicate the swap operation has completed. After SW0 released, it will execute software reset by using SCB->AIRCR register. Then this demo will run again to enter normal mode, because Demo A and B have been exchanged, the address “CODE_START” has become the start address of Demo B, Demo B will run (LED1 blinks).

```
    LCD_Display("Finished", "Restart.....");


    while (SW_Get(SW0) == 1U) {
    }

    reg_value = SCB->AIRCR;         /* software reset */
    reg_value = (uint32_t) 0x05FA0001;
    SCB->AIRCR = reg_value;
```

Flash memory operation function FLASH_EraseBlock() will erase a specified block automatically. The block is specified by parameter “block_addr”. Firstly, this function will check whether the parameter “block_addr” is illegal. Then it will use Flash driver FC_GetBlockProtectState() to check if the specified block is protected.

```
    if (ENABLE == FC_GetBlockProtectState(BlockNum)) {
        retval = FC_ERROR_PROTECTED;
    }
```

If the block is protected, it will return “FC_ERROR_PROTECTED”, otherwise it will send automatic block erase command to erase the block.

```
        *addr1 = (uint32_t) 0x000000AA; /* bus cycle 1 */
        *addr2 = (uint32_t) 0x00000055; /* bus cycle 2 */
        *addr1 = (uint32_t) 0x00000080; /* bus cycle 3 */
        *addr1 = (uint32_t) 0x000000AA; /* bus cycle 4 */
        *addr2 = (uint32_t) 0x00000055; /* bus cycle 5 */
        *BA = (uint32_t) 0x00000030;       /* bus cycle 6 */
```

Then the function will use Flash driver FC_GetBusyState() to monitor when erasing operation finished. At the same time, use a timeout counter to check if the operation is overtime.

```
        while (BUSY == FC_GetBusyState()) {       /* check if FLASH is busy with overtime counter */
            if (!(counter--)) { /* check overtime */
                retval = FC_ERROR_OVER_TIME;
                break;
            } else {
                /* Do nothing */
            }
        }
```

Function Write_Flash() will call FLASH_WritePage() to write data automatically in one page. The process of this function is basically same as FLASH_EraseBlock() except the automatic page program command.

```
*addr1 = (uint32_t) 0x000000AA; /* bus cycle 1 */
*addr2 = (uint32_t) 0x00000055; /* bus cycle 2 */
*addr1 = (uint32_t) 0x000000A0; /* bus cycle 3 */
for (i = 0U; i < FC_PAGE_SIZE; i++) {      /* bus cycle 4~67 */
      *addr3 = *source;
      source++;
}
```

7-5 GPIO

This is a simple example based on the TX03 Peripheral Driver (GPIO).

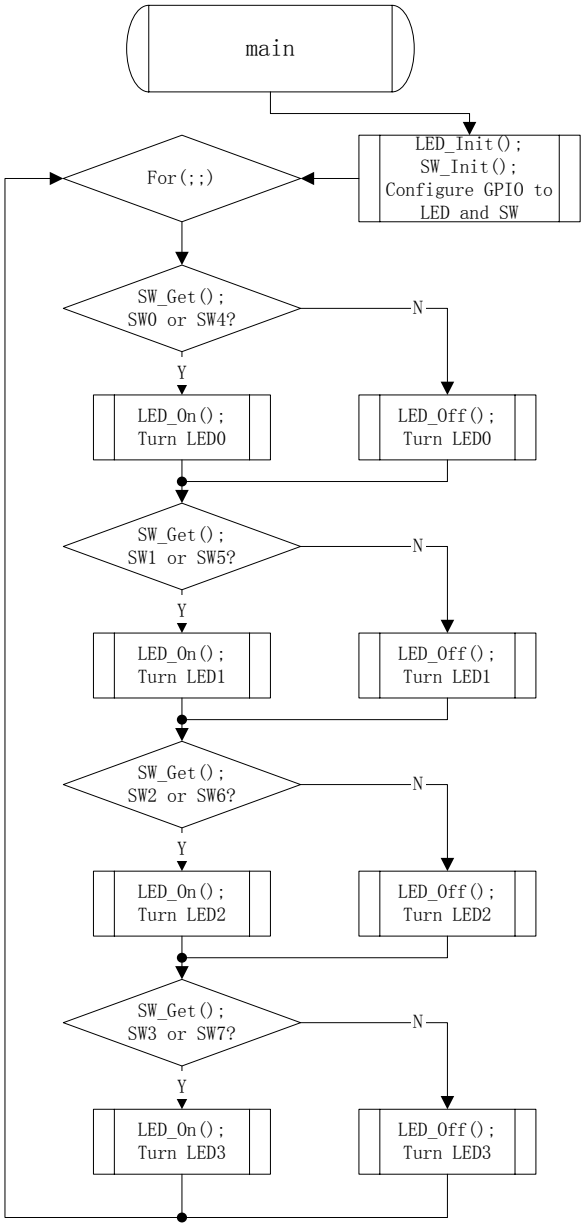
The example includes:

1. GPIO initialization
2. Write data to GPIO
3. Read data from GPIO

Connections of SW, LED, LCD:

Port	Function
PH0	SW0, SW4
PH1	SW1, SW5
PH2	SW2, SW6
PH3	SW3, SW7
PT0	LED0
PT1	LED1
PT2	LED2
PT3	LED3
PH4	SWS

- **Flow chart:**



- **Code and Explantion for the Example:**

At first, use GPIO_SetOutput(GPIO_Port GPIO_x, uint8_t Bit_x) to configure GPIO to LED, and GPIO_SetInput(GPIO_Port GPIO_x, uint8_t Bit_x) to configure GPIO to key. For example,

```
/* LED0~LED3 */
GPIO_SetOutput(LED_DATA_PORT, 0x0FU);
GPIO_WriteData(LED_DATA_PORT, 0x00U);

/* SW0~7, SWS*/
GPIO_SetInput(SW_PORT, W04_BIT|SW15_BIT|SW26_BIT|SW37_BIT);
GPIO_SetOutput(SWS_PORT, SWS_BIT);
```

In the for(; ;) process, run the LED demo: Read Switch to control LED on and LED off.

Turn on LED by using GPIO_WriteDataBit(GPIO_Port GPIO_x, uint8_t Bit_x, uint8_t BitValue).

```
uint32_t tmp;
tmp = GPIO_ReadData(LED_DATA_PORT);
tmp |= led;
GPIO_WriteData(LED_DATA_PORT, tmp);
```

Turn off LED by using GPIO_WriteDataBit(GPIO_Port GPIO_x, uint8_t Bit_x, uint8_t BitValue).

```
uint32_t tmp;
tmp = GPIO_ReadData(LED_DATA_PORT);
tmp &= ~led;
GPIO_WriteData(LED_DATA_PORT, tmp);
```

7-6 OFD

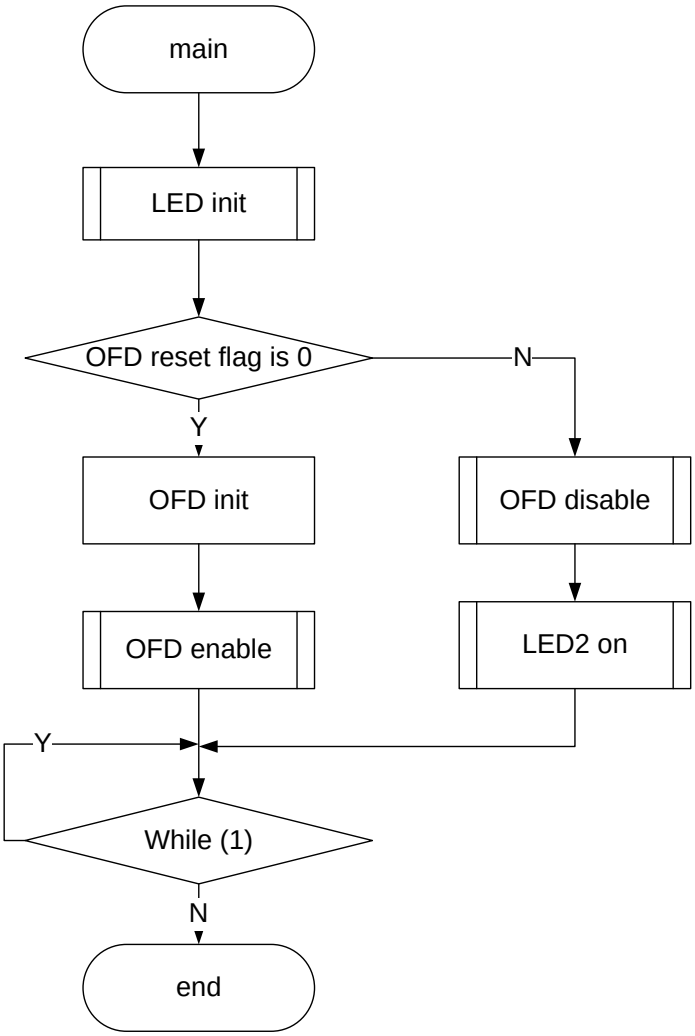
7-6-1 Example: Startup

his is a simple example based on the TX03 Peripheral Driver (OFD, CG, and GPIO).

The example includes:

1. OFD initialization (set OFD detect frequency range)
2. OFD reset flag check by CG.
3. OFD enable and disable.

- **Flowchart**



- **Code and Explanation for the Example**

At first, initialize LED.

```
Init_Led();
```

Check the OFD reset flag.

```
resetFlag = CG_GetResetFlag();
if (resetFlag.Bit.OFDReset == 0)
```

If the flag is not set, then use the external oscillation and initialize OFD to set OFD detect frequency range.

Set up the CG to use the external oscillation.

```
void ChangeToEHOSC(void)
{
    /* Use the external oscillation */
    TSB_CG->OSCCR &= 0x000FFFFFUL;
    TSB_CG->OSCCR |= 0xFFF00000UL;              /*Set the warm up time WUODR[13:2]*/
    TSB_CG_OSCCR_HOSCON = 1U;     /*Set gpio port as X1/X2 for external
```

```
oscillator */
    TSB_CG_OSCCR_XEN1 = 1U;        /*Enable external oscillator */
    TSB_CG_OSCCR_WUPSEL2 = 1U;
    TSB_CG_OSCCR_WUPSEL1 = 0U;   /*Select warm-up clock */
    TSB_CG_OSCCR_WUEON = 1U;      /*Start warm up */
    while (TSB_CG_OSCCR_WUEF) {}/* Warm-up */
    TSB_CG_OSCCR_OSCSEL = 1U;     /*Use the external oscillation */
    TSB_CG_OSCCR_XEN2 = 1U;        /*Enable internal oscillator for ofd */
}
```

Then set detection frequency.

```
    OFD_SetDetectionFrequency(OFD_HIGHER_COUNT,
                                                    OFD_LOWER_COUNT);
```

If define DEMO2, the abnormal frequency range is set.

```
    OFD_SetDetectionFrequency(OFD_HIGHER_COUNT_ABNORMAL,
                                                OFD_LOWER_COUNT_ABNORMAL);
```

Enable OFD after initialization.

```
    OFD_Enable();
```

After above setting, OFD will detect the frequency of external clock. If the clock exceeds the detection frequency range (disturb the oscillator or define DEMO2), OFD will generate a reset for I/O. Then software will be reset. The OFD reset flag will be set.
If the flag is detected, OFD will be disabled and LED will be turned on.

```
    OFD_Disable();
    OFDReset = 1U;
```

Led1 and Led2 will should out the system clock status. If it use external high speed oscillator the led1 will blinking, else if it use internal high speed clock, the led2 will blinking.

```
    while (1U) {
        if (OFDReset == 1U) {
            LED_On(LED2);
            Delay();
            LED_Off(LED2);
            Delay();
        } else {
            LED_On(LED1);
            Delay();
            LED_Off(LED1);
            Delay();
        }
    }
```

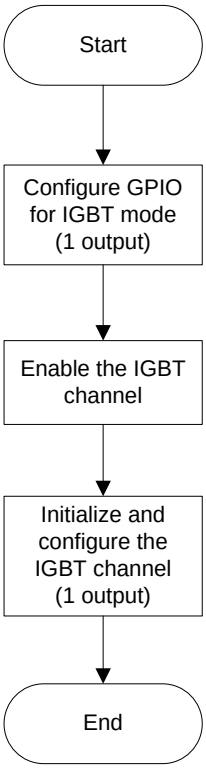
7-7 IGBT

7-7-1 Example: Single PPG Output

This is a simple example based on the TX03 Peripheral Driver (IGBT and GPIO).

The example includes:
1. IGBT timer's configuration and initialization (one PPG output).
2. IGBT timer's EMG protection function.
3. The usage of IGBT time's status and EMG interrupt.

- **Flowchart**



- **Code and Explanation for the Example**

After the LED configuration, create IGBT_InitTypeDef structure and fill all the data fields. For example,

```
IGBT_InitTypeDef myIGBT;
/* IGBT trigger start: falling edge start and active level is "Low" */
myIGBT.StartMode = IGBT_FALLING_TRG_START;
/* IGBT operation: continuous operation */
myIGBT.OperationMode = IGBT_CONTINUOUS_OUTPUT;
```

```
    /* IGBT stopping status: initial output satus and counter */
    myIGBT.CntStopState = IGBT_OUTPUT_INACTIVE;
    /* Trigger edge accept mode: Don't accept trigger during active level */
    myIGBT.ActiveAcceptTrg = DISABLE;
    /* Interrupt cycle: Every one cycle */
    myIGBT.INTPeriod = IGBT_INT_PERIOD_1;
    /* fperiph = fc = 10MHz*4 = 40MHz, T0 = fperiph = 40MHz, figbt = T0/2 = 20MHz */
    myIGBT.ClkDiv = IGBT_CLK_DIV_2;
    /* MTOUT00 initial state is Low, and active level is High */
    myIGBT.Output0Init = IGBT_OUTPUT_HIGH_ACTIVE;
    /* Disable MTOUT10 output */
    myIGBT.Output1Init = IGBT_OUTPUT_DISABLE;
    /* Trigger input noise elimination time: 240/fsys */
    myIGBT.TrgDenoiseDiv = IGBT_DENOISE_DIV_240;
    myIGBT.Output0ActiveTiming = 1U;
    myIGBT.Output0InactiveTiming = 1U + IGBT_PPG_PERIOD_50US / 2;
    /* Period: 50us */
    myIGBT.Period = IGBT_PPG_PERIOD_50US;
    /* The polarity of MTOUT0x at EMG protection: High-impedance */
    myIGBT.EMGFunction = IGBT_EMG_OUTPUT_HIZ;
    /* EMG input noise elimination time: 240/fsys */
    myIGBT.EMGDenoiseDiv = IGBT_DENOISE_DIV_240;
```

Then enable the IGBT channel and EMG protection interrupt, and cancel EMG state before the operation of initialization and configuration.

```
    /* Enable IGBT and EMG interrupt */
    IGBT_Enable(IGBT0);
    NVIC_EnableIRQ(INTMTEMG0_IRQn);

    /* If the timer is still running, wait until it stops */
    do {
        counter_state = IGBT_GetCntState(IGBT0);
    } while (counter_state == BUSY);
    /* Cancel the EMG protection state */
    do {
        cancel_result = IGBT_CancelEMGState(IGBT0);
    } while (cancel_result == ERROR);
```

If IGBT_CancelEMGState() returns **SUCCESS**, call IGBT_Init(), the initialization and configuration of IGBT can be complete.

```
  IGBT_Init(IGBT0, &myIGBT);
```

Before accepting the start trigger, the software start command must be issued at first.

```
IGBT_SetSWRunState(IGBT0, IGBT_RUN);
```

After this, the corresponding port will output PPG wave once the start trigger is detected.

If EMG protection level on GEMG is active, the EMG interrupt occurs. Turn on the LED1 and stop the timer.

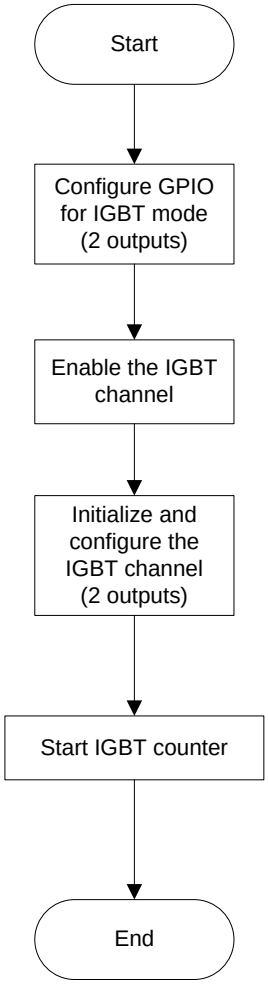
```
void INTMTEMG0_IRQHandler(void)
{
    /* If EMG protection, turn on the LED and stop the timer */
    LED_On(LED1);
    IGBT_SetSWRunState(IGBT0, IGBT_STOP);
}
```

7-7-2 Example: Double PPG Output

This is a simple example based on the TX03 Peripheral Driver (IGBT and GPIO).

The example includes:
1. IGBT timer's configuration and initialization (two PPG output).
2. IGBT timer's EMG protection function.
3. The usage of IGBT time's status and EMG interrupt.

- **Flowchart**



- **Code and Explanation for the Example**

After the LED and switch port configuration, create IGBT_InitTypeDef structure and fill all the data fields. For example,

```
IGBT_InitTypeDef myIGBT;
/* IGBT trigger start: command start */
myIGBT.StartMode = IGBT_CMD_START;
/* IGBT operation: continuous operation */
myIGBT.OperationMode = IGBT_CONTINUOUS_OUTPUT;
/* IGBT stopping status: initial output satus and counter */
myIGBT.CntStopState = IGBT_OUTPUT_INACTIVE;
/* Trigger edge accept mode: Don't accept trigger during active level */
myIGBT.ActiveAcceptTrg = DISABLE;
/* Interrupt cycle: Every one cycle */
myIGBT.INTPeriod = IGBT_INT_PERIOD_1;
/* fperiph = fc = 10MHz*4 = 40MHz, T0 = fperiph = 40MHz, figbt = T0/2 = 20MHz */
myIGBT.ClkDiv = IGBT_CLK_DIV_2;
```

```
    /* MTOUT00 initial state is Low, and active level is High */
    myIGBT.Output0Init = IGBT_OUTPUT_HIGH_ACTIVE;
    /* MTOUT10 initial state is Low, and active level is High */
    myIGBT.Output1Init = IGBT_OUTPUT_HIGH_ACTIVE;
    /* Trigger input noise elimination time: no use */
    myIGBT.TrgDenoiseDiv = IGBT_NO_DENOISE;
    myIGBT.Output0ActiveTiming = 1U;
    myIGBT.Output0InactiveTiming = 1U + IGBT_ACTIVE_PERIOD_20US;
    myIGBT.Output1ActiveTiming = myIGBT.Output0InactiveTiming +
                                 IGBT_DEAD_TIME_5US;
    myIGBT.Output1InactiveTiming = myIGBT.Output1ActiveTiming +
                                   IGBT_ACTIVE_PERIOD_20US;
    /* Period: 50us */
    myIGBT.Period = IGBT_PPG_PERIOD_50US;
    /* The polarity of MTOUT0x at EMG protection: High-impedance */
    myIGBT.EMGFunction = IGBT_EMG_OUTPUT_HIZ;
    /* EMG input noise elimination time: 240/fsys */
    myIGBT.EMGDenoiseDiv = IGBT_DENOISE_DIV_240;
```

Then enable the IGBT channel and EMG protection interrupt, and cancel EMG state before the operation of initialization and configuration.

```
    /* Enable IGBT and EMG interrupt */
    IGBT_Enable(IGBT0);
    NVIC_EnableIRQ(INTMTEMG0_IRQn);

    /* If the timer is still running, wait until it stops */
    do {
        counter_state = IGBT_GetCntState(IGBT0);
    } while (counter_state == BUSY);
    /* Cancel the EMG protection state */
    do {
        cancel_result = IGBT_CancelEMGState(IGBT0);
    } while (cancel_result == ERROR);
```

If IGBT_CancelEMGState() returns **SUCCESS**, call IGBT_Init(), the initialization and configuration of IGBT can be complete.

```
  IGBT_Init(IGBT0, &myIGBT);
```

If switch is ON, send the start command to run.

```
  /* If switch is ON, start to run IGBT timer */
  if (SWITCH_ON) {
      IGBT_SetSWRunState(IGBT0, IGBT_RUN);
  } else {
```

```
  /* Do nothing */
}
```

If EMG protection level on GEMG is active, the EMG interrupt occurs. Turn on the LED1 and stop the timer.

```
void INTMTEMG0_IRQHandler(void)
{
      /* If EMG protection, turn on the LED and stop the timer */
      LED_On(LED1);
      IGBT_SetSWRunState(IGBT0, IGBT_STOP);
}
```

7-8 RMC

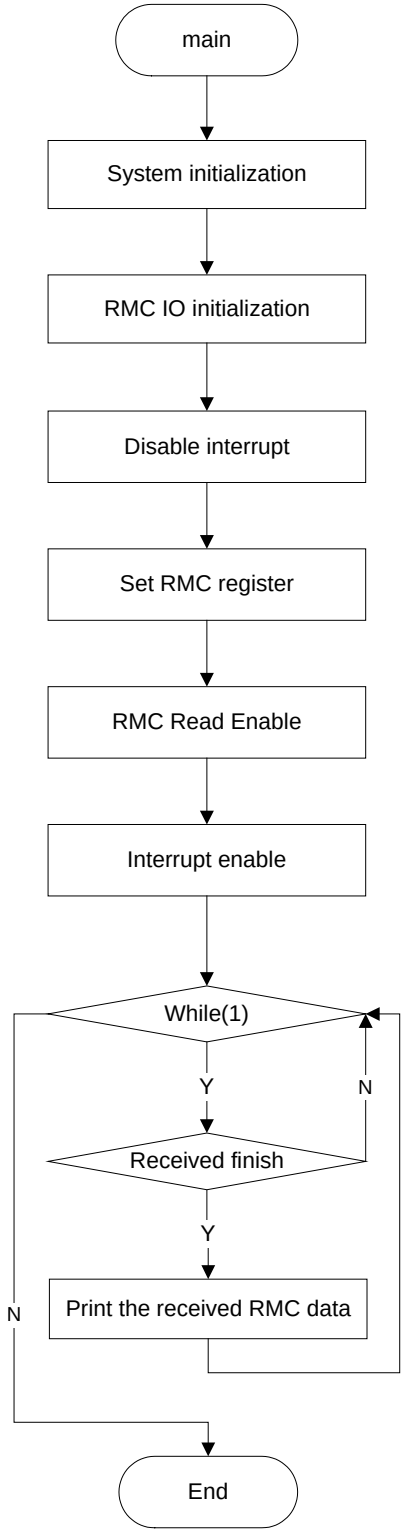
7-8-1 Example: RMC receiving

This is a simple example based on the TX03 Peripheral Driver (RMC, UART, GPIO).

The example includes:

1. RMC IO initialization
2. Receiving RMC data

- **Flow Chart**


main
System initialization
RMC IO initialization
Disable interrupt
Set RMC register
RMC Read Enable
Interrupt enable
While(1)
Y
N
Received finish
Y
N
Print the received RMC data
End

- **Code and Explanation for the Example**

At first, in main(), create a myRMC structure, then fill all the data fields.
For example

```
RMC_InitTypeDef myRMC;

myRMC.LeaderPara.MaxCycle = RMC_MAX_CYCLE;
myRMC.LeaderPara.MinCycle = RMC_MIN_CYCLE;
myRMC.LeaderPara.MaxLowWidth = RMC_MAX_LOW_WIDTH;
myRMC.LeaderPara.MinLowWidth = RMC_MIN_LOW_WIDTH;
myRMC.LeaderPara.LeaderDetectionState = ENABLE;
myRMC.LeaderPara.LeaderINTState = DISABLE;
myRMC.FallingEdgeINTState = DISABLE;
myRMC.SignalRxMethod = RMC_RX_IN_CYCLE_METHOD;
myRMC.LowWidth = RMC_TRG_LOW_WIDTH;
myRMC.MaxDataBitCycle = RMC_TRG_MAX_DATA_BIT_CYCLE;
myRMC.LargerThreshold = RMC_LARGER_THRESHOLD;
myRMC.SmallerThreshold = RMC_SMALLER_THRESHOLD;
myRMC.InputSignalReversedState = DISABLE;
myRMC.NoiseCancellationTime = RMC_NOISE_CANCELLATION_TIME;
```

Then, enable and initialize the RMC channel 0.

```
RMC_Enable(TSB_RMC0);
```

Initial the specified RMC channel with the structure which includes the basic RMC configuration.

```
RMC_Init(TSB_RMC0, &myRMC);
```

Enable the reception of the specified RMC0 channel.

```
RMC_SetRxCtrl(TSB_RMC0, ENABLE);
```

In interrupt INTRMCRX_IRQHandler():
Get the interrupt factor for the specified RMC0 channel.

```
RMC_INTFactor myRMC_INTFactor;
myRMC_INTFactor = RMC_GetINTFactor(TSB_RMC0);
```

Get the leader detection result for the specified RMC0 channel.

```
RMC_LeaderDetection myRMC_LeaderDetection;
myRMC_LeaderDetection = RMC_GetLeader(TSB_RMC0);
```

Get the received data from the specified RMC0 channel.

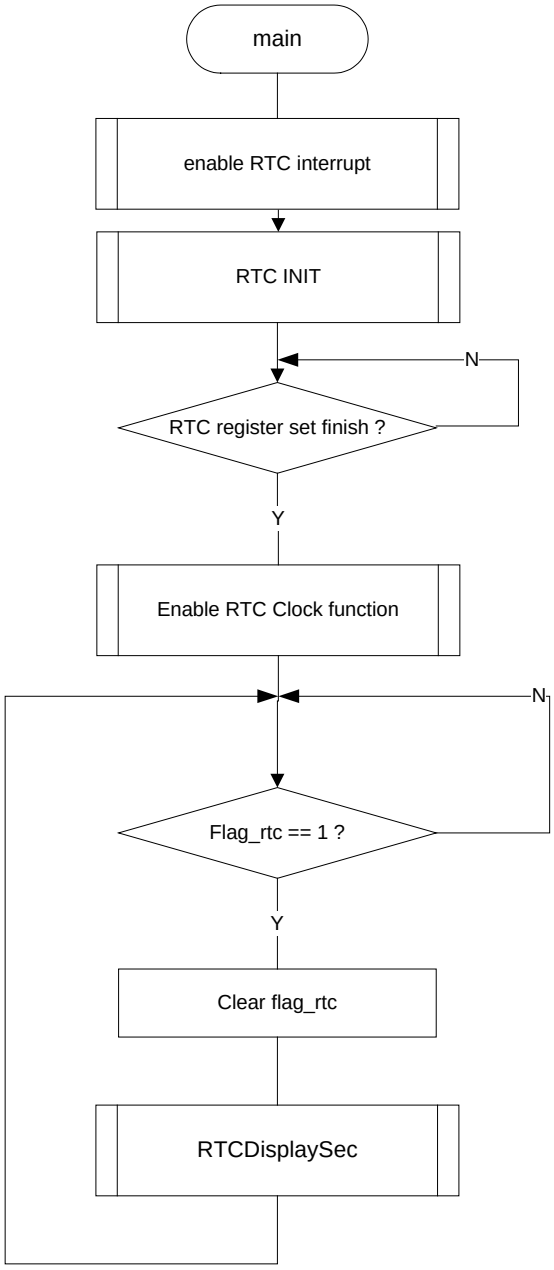
```
RMC_RxDataTypeDef myRMC_RxDataDef;
myRMC_RxDataDef = RMC_GetRxData(TSB_RMC0);
```

7-9 RTC

This is a simple example based on the TX03 Peripheral Driver (RTC, CG, GPIO).

The example includes:
1. RTC initialization
2. Get RTC date and time

- **Flow chart:**



- **Code and Explanation for the Example:**

At first set source(RTC interrupt) to exit sleep mode

```
CG_SetSTBYReleaseINTSrc(CG_INT_SRC_RTC,
                        CG_INT_ACTIVE_STATE_FALLING, ENABLE);
```

Initialize RTC .Create a RTC_DateTypeDef and RTC_TimeTypeDef structure and fill all the data fields. For example, Initial setting: 2010/10/22 12:50:55, 24hours format.

```
Date_Struct.LeapYear = RTC_LEAP_YEAR_2;
Date_Struct.Year = (uint8_t) 10U;
Date_Struct.Month = (uint8_t) 10U;
Date_Struct.Date = (uint8_t) 22U;
Date_Struct.Day = RTC_FRI;
```

```
Time_Struct.HourMode = RTC_24_HOUR_MODE;
Time_Struct.Hour = (uint8_t) 12U;
Time_Struct.Min = (uint8_t) 50U;
Time_Struct.Sec = (uint8_t) 55U;
```

Disable clock and alarm function.

```
RTC_DisableClock();
RTC_DisableAlarm();
```

Reset RTC sec counter, enable 1Hz interrupt, enable RTCINT.

```
RTC_ResetClockSec();
RTC_SetAlarmOutput(RTC_PULSE_1_HZ);
RTC_SetRTCINT(ENABLE);
```

Set RTC time and date value

```
RTC_SetTimeValue(&Time_Struct);
RTC_SetDateValue(&Date_Struct);
```

After the setting above, enable RTC interrupt. Waiting for RTC register set finished , then enable RTC Clock function.

```
NVIC_EnableIRQ(INTRTC_IRQn);

/* waiting for RTC register set finish */
while (fRTC_1HZ_INT != 1U) {
      /* Do nothing */
}
fRTC_1HZ_INT = 0U;

/* Enable RTC Clock function */
RTC_EnableClock();
```

In RTC interrupt request function,The RTC interrupt will occurs a interrupt flag every second. Then RTC interrupt request will be cleared.

```
fRTC_1HZ_INT = 1U;
CG_ClearINTReq(CG_INT_SRC_RTC);
```

After interrupt flag occus RTC ones place of second value will be sent to LED.
Following is shown how to get RTC date and time value.

```
Sec = RTC_GetSec();
tmp = Sec % 10U;
SegArrayDisplay(ledchar[tmp])
```

7-10 SBI

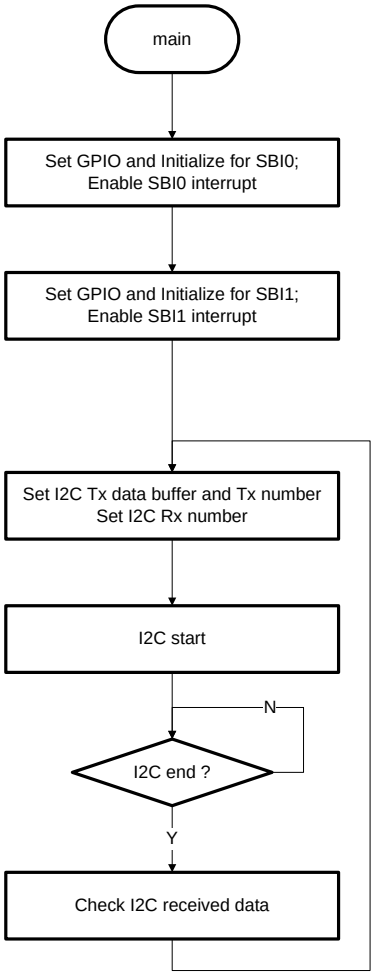
This is a simple example based on the TX03 Peripheral Driver (SBI, GPIO).

The example includes:

1. SBI configuration and I2C initialization
2. I2C master send data process

3. I2C slave receive data process

- **Flow Chart**



- **Code and Explanation for the Example**

At first, configure GPIO for SBI0 and SBI1 I2C mode.

```
SBI0_IO_Configuration();
SBI1_IO_Configuration();
```

Initialize and configure SBI0 channel and enable INTSBI0.

```
myI2C.I2CSelfAddr = SELF_ADDR;
myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2C.I2CACKState = ENABLE;
myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
SBI_Enable(TSB_SBI0);
SBI_SWReset(TSB_SBI0);
SBI_InitI2C(TSB_SBI0, &myI2C);
NVIC_EnableIRQ(INTSBI0_IRQn);
```

Then enable, initialize and configure SBI1 channel and enable INTSBI1.

```
myI2C.I2CSelfAddr = SLAVE_ADDR;
myI2C.I2CDataLen = SBI_I2C_DATA_LEN_8;
myI2C.I2CACKState = ENABLE;
myI2C.I2CClkDiv = SBI_I2C_CLK_DIV_328;
SBI_Enable(TSB_SBI1);
```

```
    SBI_SWReset(TSB_SBI1);
    SBI_InitI2C(TSB_SBI1, &myI2C);
    NVIC_EnableIRQ(INTSBI1_IRQn);
```

After the above setting, start I2C read.
Clear I2C Rx buffer, initialize the SBI Tx buffer and length, and clear Rx buffer.

```
        /* Initialize TRx buffer and Tx length */
    case MODE_SBI_I2C_INITIAL:
        gI2CTxDataLen = 7U;
        gI2CTxData[0] = gI2CTxDataLen;
        gI2CTxData[1] = 'T';
        gI2CTxData[2] = 'O';
        gI2CTxData[3] = 'S';
        gI2CTxData[4] = 'H';
        gI2CTxData[5] = 'I';
        gI2CTxData[6] = 'B';
        gI2CTxData[7] = 'A';
        gI2CWCnt = 0U;
        for (gICnt = 0U; gICnt < 8U; gICnt++) {
            gI2CRxData[gICnt] = 0U;
        }
        gSBIMode = MODE_SBI_I2C_START;
        break;
```

Check if the I2C bus is free or not, SBI_SetSendData() is used to set data “SLAVE_ADDR”.and direction is “SBI_I2C_SEND” to SBI data buffer; then SBI_GenerateI2CStart(TSB_SBI0) is used to to start I2C process.

```
        /* Check I2C bus state and start TRx */
    case MODE_SBI_I2C_START:
        i2c_state = SBI_GetI2CState(TSB_SBI0);
        if (!i2c_state.Bit.BusState) {
            SBI_SetSendData(TSB_SBI0, SLAVE_ADDR | SBI_I2C_SEND);
            SBI_GenerateI2CStart(TSB_SBI0);
            gSBIMode = MODE_SBI_I2C_TRX;
        } else {
            /* Do nothing */
        }
        break;
```

The data transfer is handled in INTSBI0.
The data receive is handled in INTSBI1.

In INTSBI0 function, I2C master send process is handled according to I2C bus state. During I2C master sending, SBI_SetSendData() is used to send next data, SBI_GenerateI2CStop() is used to stop I2C when I2C process is finished.

```
void INTSBI0_IRQHandler(void)
{
    TSB_SBI_TypeDef *SBIx;
    SBI_I2CState sbi_sr;

    SBIx = TSB_SBI0;
    sbi_sr = SBI_GetI2CState(SBIx);

    if (sbi_sr.Bit.MasterSlave) {           /* Master mode */
        if (sbi_sr.Bit.TRx) {     /* Tx mode */
            if (sbi_sr.Bit.LastRxBit) { /* LRB=1: the receiver requires no further data. */
                SBI_GenerateI2CStop(SBIx);
            } else {                    /* LRB=0: the receiver requires further data. */
                if (gI2CWCnt <= gI2CTxDataLen) {
                    SBI_SetSendData(SBIx, gI2CTxData[gI2CWCnt]);
```

```
                                                                    /* Send next data */
                        gI2CWCnt++;
                    } else {            /* I2C data send finished. */
                        SBI_GenerateI2CStop(SBIx);    /* Stop I2C */
                    }
                }
            } else {                    /* Rx Mode */
                /* Do nothing */
            }
        } else {                        /* Slave mode */
            /* Do nothing */
        }
}
```

In INTSBI1 function, I2C slave receive process is handled according to I2C bus state, SBI_GetReceiveData() is used to read received data in SBI buffer, I2C stop condition is controlled by master.

```
void INTSBI1_IRQHandler(void)
{
    uint32_t tmp = 0U;
    TSB_SBI_TypeDef *SBIy;
    SBI_I2CState sbi1_sr;

    SBIy = TSB_SBI1;
    sbi1_sr = SBI_GetI2CState(SBIy);

    if (!sbi1_sr.Bit.MasterSlave) {         /* Slave mode */
        if (!sbi1_sr.Bit.TRx) { /* Rx Mode */
            if (sbi1_sr.Bit.SlaveAddrMatch) {
                /* First read is dummy read for Slave address recognize */
                tmp = SBI_GetReceiveData(SBIy);
                gI2CRCnt = 0U;
            } else {
                /* Read I2C received data and save to I2C_RxData buffer */
                tmp = SBI_GetReceiveData(SBIy);
                gI2CRxData[gI2CRCnt] = tmp;
                gI2CRCnt++;
            }
        } else {                        /* Tx Mode */
            /* Do nothing */
        }
    } else {                            /* Master mode */
        /* Do nothing */
    }
}
```

7-11 SIO/UART

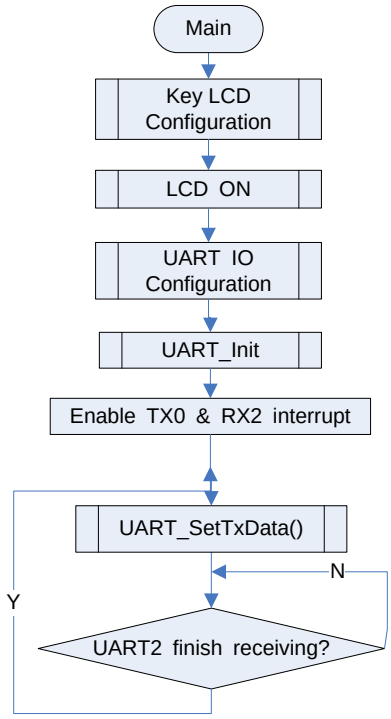
7-11-1 Example: Transmission from UART0 to UART2

This is a simple example based on the TX03 Peripheral Driver (UART, GPIO).

The example includes:
1. UART configuration and initialization
2. UART TRX process

3. Use UART0 TX interrupt to send data
4. Use UART2 RX interrupt to receive data that from UART0

- **Flowchart**



- **Code and Explanation for the Example**

At first configure the GPIO and initialize UART.

Use GPIO peripheral drivers configure GPIO for UART0.

```
GPIO_SetOutputEnableReg(GPIO_PE, GPIO_BIT_0, ENABLE);
GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_0);
GPIO_EnableFuncReg(GPIO_PE, GPIO_FUNC_REG_1, GPIO_BIT_1);
GPIO_SetInputEnableReg(GPIO_PE, GPIO_BIT_1, ENABLE);
```

Configure GPIO for UART2.

```
GPIO_SetOutputEnableReg(GPIO_PF, GPIO_BIT_0, ENABLE);
GPIO_EnableFuncReg(GPIO_PF, GPIO_FUNC_REG_1, GPIO_BIT_0);
GPIO_EnableFuncReg(GPIO_PF, GPIO_FUNC_REG_1, GPIO_BIT_1);
GPIO_SetInputEnableReg(GPIO_PF, GPIO_BIT_1, ENABLE);
```

Create a UART_InitTypeDef structure and fill all the data fields. For example,

```
UART_InitTypeDef myUART;

myUART.BaudRate = 115200U;    /* baut rate = 115200 */
myUART.DataBits = UART_DATA_BITS_8; /* no handshake, 8-bit data, clock by baud rate generator */
myUART.StopBits = UART_STOP_BITS_1; /* 1-bit stop, LSB,    W-buff enable */
myUART.Parity = UART_NO_PARITY;
myUART.Mode = UART_ENABLE_RX | UART_ENABLE_TX;
myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
```

Use UART peripheral drivers to enable and initialize UART0/2 channels.

```
UART_Enable(UART0);
UART_Init(UART0, &myUART);

UART_Enable (UART2);
UART_Init(UART2, &myUART);
```

After above setting, enable UART0 TX interrupt and UART2 RX interrupt.

```
NVIC_EnableIRQ(INTTX0_IRQn);
NVIC_EnableIRQ(INTRX2_IRQn);
```

Then start sending data. Here TxBuf is a character array.

```
UART_SetTxData(UART0, TxBuf[0]);
```

The rest process of data flow is finished in ISR of UART0 TX interrupt routine and UART2 RX interrupts routine.

UART0 TX interrupt routine:

```
void INTTX0_IRQHandler(void)
{
	volatile UART_Err err;

	if (TxCounter < NumToBeTx) {
		UART_SetTxData(UART0, TxBuffer[TxCounter++]);
	} else {
		err = UART_GetErrState(UART0);
	}
}
```

UART2 RX interrupt routine:

```
void INTRX2_IRQHandler(void)
{
	volatile UART_Err err;

	err = UART_GetErrState(UART2);
	if (UART_NO_ERR == err) {
		RxBuffer[RxCounter++] = (uint8_t) UART_GetRxData(UART2);
	}
}
```

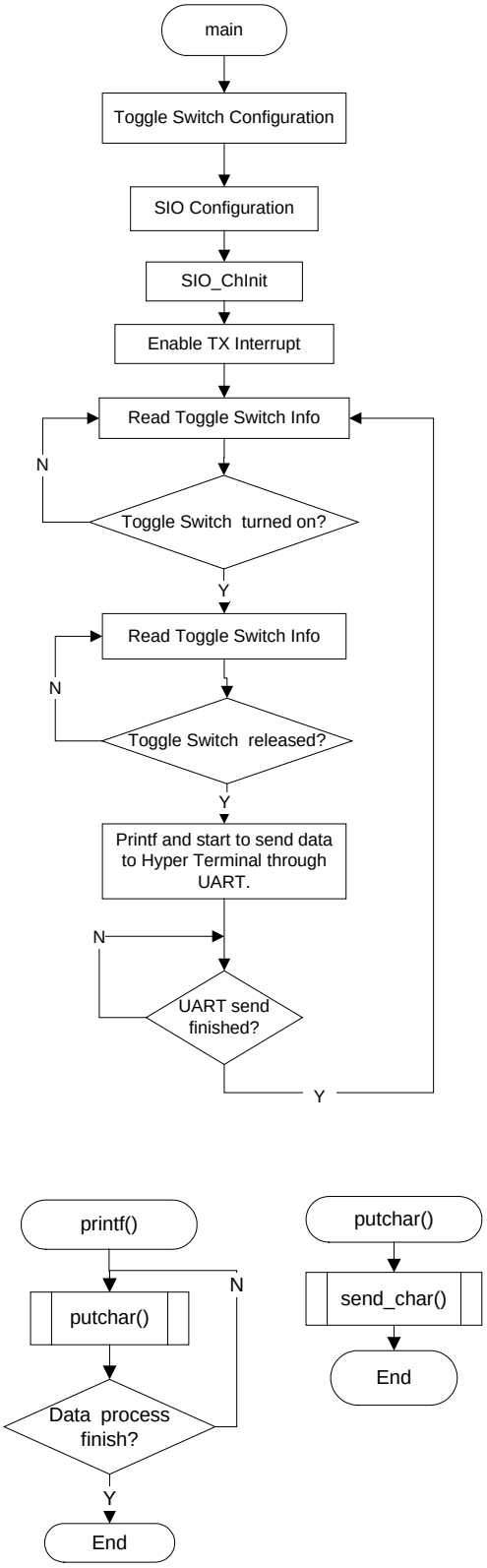
7-11-2 Example: Retarget

This is a simple example based on the TX03 Peripheral Driver (UART, GPIO).

The example includes:

1. UART configuration and initialization.
2. UART TX process.
3. Use UART1 TX interrupt to send data.
4. Retarget printf() to UART1.

- **Flowchart**



- **Code and Explanation for the Example**

At first configure the GPIO and initialize UART.

Use GPIO peripheral drivers configure GPIO for UART.

```
    GPIO_SetOutputEnableReg(GPIO_PC, GPIO_BIT_0, ENABLE);
```

```
    GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_1, GPIO_BIT_0);
    GPIO_EnableFuncReg(GPIO_PC, GPIO_FUNC_REG_1, GPIO_BIT_1);
    GPIO_SetInputEnableReg(GPIO_PC, GPIO_BIT_1, ENABLE);
    GPIO_SetPullUp(GPIO_PC, GPIO_BIT_0, ENABLE);
    GPIO_SetPullUp(GPIO_PC, GPIO_BIT_1, ENABLE);
```

Create a UART_InitTypeDef structure and fill all the data fields. For example,

```
    UART_InitTypeDef myUART;

    /* configure SIO1 for reception */
    UART_Enable(UART_RETARGET);
    myUART.BaudRate = 115200U;    /* baud rate = 115200 */
    myUART.DataBits = UART_DATA_BITS_8; /* no handshake, 8-bit data, clock by
baud rate generator */
    myUART.StopBits = UART_STOP_BITS_1; /* 1-bit stop, LSB,    W-buff enable */
    myUART.Parity = UART_NO_PARITY;
    myUART.Mode = UART_ENABLE_TX;
    myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
    UART_Init(UART_RETARGET, &myUART);
```

After above setting, enable UART TX interrupt.

```
    NVIC_EnableIRQ(RETARGET_INT)
```

Then start sending data. Here TxBuffer is a character array.

```
    printf("%s\r\n", TxBuffer);
```

The rest process of data flow is finished in ISR of UART1 TX interrupt routine
UART1 TX interrupt routine:

```
void INTTX1_IRQHandler(void)
{
    if (gSIORdIndex < gSIOWrIndex) {       /* buffer is not empty */
        UART_SetTxData(UART_RETARGET,          gSIOTxBuffer[gSIORdIndex++]);
/* send data */
        fSIO_INT = SET;              /* SIO1 INT is enable */
    } else {
        /* disable SIO1 INT */
        fSIO_INT = CLEAR;
        NVIC_DisableIRQ(RETARGET_INT);
        fSIOTxOK = YES;
    }
    if (gSIORdIndex >= gSIOWrIndex) {     /* reset buffer index */
        gSIOWrIndex = CLEAR;
        gSIORdIndex = CLEAR;
    } else {
        /* Do nothing */
    }
}
```

Function printf() will call putchar() for IAR Compiler and fputc() for RealView Compiler to output data to UART.

```
#if defined ( __CC_ARM     )          /* RealView Compiler */
struct __FILE {
    int handle;                       /* Add whatever you need here */
};
FILE __stdout;
FILE __stdin;
int fputc(int ch, FILE * f)
#elif defined ( __ICCARM__ )       /*IAR Compiler */
int putchar(int ch)
```

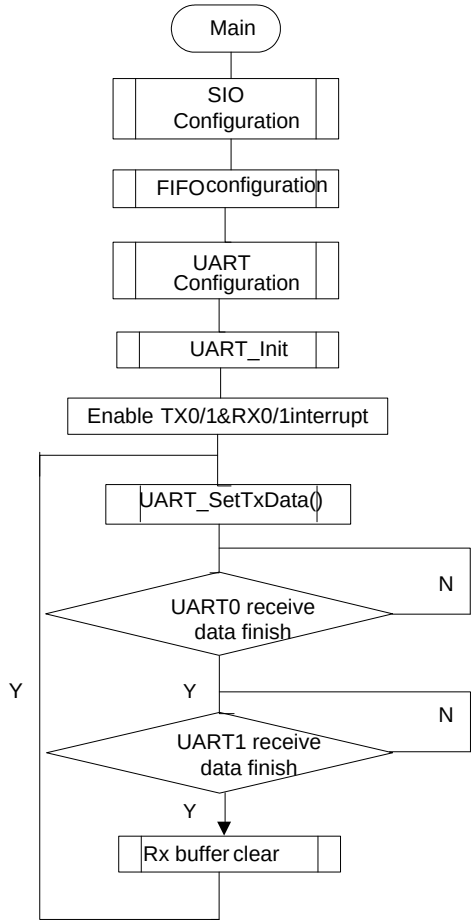
```
#endif
{
	return (send_char(ch));
}

uint8_t send_char(uint8_t ch)
{
	while (gSIORdIndex != gSIOWrIndex) {		/* wait for finishing sending */
		/* do nothing */
	}
	gSIOTxBuffer[gSIOWrIndex++] = ch;	/* fill TxBuffer */
	if (fSIO_INT == CLEAR) {	/* if SIO INT disable, enable it */
		fSIO_INT = SET;		/* set SIO INT flag */
		UART_SetTxData(UART_RETARGET, gSIOTxBuffer[gSIORdIndex++]);
		NVIC_EnableIRQ(RETARGET_INT);
	}

	return ch;
}
```

7-11-3 Example: UART FIFO

This is a simple example based on the TX03 Peripheral Driver (UART, GPIO).

The example includes:
UART and FIFO configuration and initialization.
UART send and receive data use FIFO process.

- **Flowchart**



- **Code and Explanation for the Example**

At first configure the GPIO and initialize UART.

Use GPIO peripheral drivers configure GPIO for UART0 and UART1.

```
void SIO_Configuration(TSB_SC_TypeDef * SCx)
{
      if (SCx == TSB_SC0) {
            TSB_PE->CR |= GPIO_BIT_0;
            TSB_PE->FR1 |= GPIO_BIT_0;
            TSB_PE->FR1 |= GPIO_BIT_1;
            TSB_PE->IE |= GPIO_BIT_1;
      } else if (SCx == TSB_SC1) {
            TSB_PC->CR |= GPIO_BIT_0;
            TSB_PC->FR1 |= GPIO_BIT_0;
            TSB_PC->FR1 |= GPIO_BIT_1;
            TSB_PC->IE |= GPIO_BIT_1;
      }
}
```

Create a UART_InitTypeDef structure and fill all the data fields. For example,

```
      UART_InitTypeDef myUART;

    /* configure SIO0 for reception */
    UART_Enable(UART_RETARGET);
    myUART.BaudRate = 115200U;    /* baud rate = 115200 */
```

```
    myUART.DataBits = UART_DATA_BITS_8; /* no handshake, 8-bit data, clock by baud rate generator */
    myUART.StopBits = UART_STOP_BITS_1; /* 1-bit stop, LSB,    W-buff enable */
    myUART.Parity = UART_NO_PARITY;
    myUART.Mode = UART_ENABLE_TX|UART_ENABLE_RX;
    myUART.FlowCtrl = UART_NONE_FLOW_CTRL;
    UART_Init(UART_RETARGET, &myUART);
```

Use UART peripheral drivers to enable and initialize UART0/1 channels.

```
     UART_Enable(UART0);
     UART_Init(UART0, &myUART);

     UART_Enable(UART1);
     UART_Init(UART1, &myUART);
```

FIFO configuration.

```
    UART_RxFIFOByteSel(UART0,UART_RXFIFO_RXFLEVEL);
    UART_RxFIFOByteSel(UART1,UART_RXFIFO_RXFLEVEL);

    UART_TxFIFOINTCtrl(UART0,ENABLE);
    UART_TxFIFOINTCtrl(UART1,ENABLE);

    UART_RxFIFOINTCtrl(UART0,ENABLE);
    UART_RxFIFOINTCtrl(UART1,ENABLE);

    UART_TRxAutoDisable(UART0,UART_RXTXCNT_AUTODISABLE);
    UART_TRxAutoDisable(UART1,UART_RXTXCNT_AUTODISABLE);

    UART_FIFOConfig(UART0,ENABLE);
    UART_FIFOConfig(UART1,ENABLE);

    UART_RxFIFOFillLevel(UART0, UART_RXFIFO4B_FLEVLE_4_2B);
    UART_RxFIFOFillLevel(UART1, UART_RXFIFO4B_FLEVLE_4_2B);

    UART_RxFIFOINTSel(UART0,UART_RFIS_REACH_EXCEED_FLEVEL);
    UART_RxFIFOINTSel(UART1,UART_RFIS_REACH_EXCEED_FLEVEL);

    UART_RxFIFOClear(UART0);
    UART_RxFIFOClear(UART1);

    UART_TxFIFOFillLevel(UART0, UART_TXFIFO4B_FLEVLE_0_0B);
    UART_TxFIFOFillLevel(UART1, UART_TXFIFO4B_FLEVLE_0_0B);

    UART_TxFIFOINTSel(UART0,UART_TFIS_REACH_NOREACH_FLEVEL);
    UART_TxFIFOINTSel(UART1,UART_TFIS_REACH_NOREACH_FLEVEL);

    UART_TxFIFOClear(UART0);
    UART_TxFIFOClear(UART1);
```

After above setting, enable UART0/1 interrupt.

```
    NVIC_EnableIRQ(INTTX0_IRQn);
    NVIC_EnableIRQ(INTRX1_IRQn);

    NVIC_EnableIRQ(INTTX1_IRQn);
    NVIC_EnableIRQ(INTRX0_IRQn);
```

The rest process of data flow is finished in ISR of UART0/1 RX and TX interrupt routine
UART0 TX interrupt routine:

```
void INTTX0_IRQHandler(void)
```

```
{
	volatile UART_Err err;

	if (TxCounter < NumToBeTx) {
		UART_SetTxData(UART0, TxBuffer[TxCounter++]);
	} else {
		err = UART_GetErrState(UART0);
	}
}
```

UART1 TX interrupt routine:

```
void INTTX1_IRQHandler(void)
{
	volatile UART_Err err;

	if (TxCounter1 < NumToBeTx1) {
		UART_SetTxData(UART1, TxBuffer1[TxCounter1++]);
	} else {
		err = UART_GetErrState(UART1);
	}
}
```

UART0 RX interrupt routine:

```
void INTRX0_IRQHandler(void)
{
	volatile UART_Err err;

	err = UART_GetErrState(UART0);
	if (UART_NO_ERR == err) {
		RxBuffer[RxCounter++] = (uint8_t) UART_GetRxData(UART0);
	}
}
```

UART1 RX interrupt routine:

```
void INTRX1_IRQHandler(void)
{
	volatile UART_Err err;

	err = UART_GetErrState(UART1);
	if (UART_NO_ERR == err) {
		RxBuffer1[RxCounter1++] = (uint8_t) UART_GetRxData(UART1);
	}
}
```

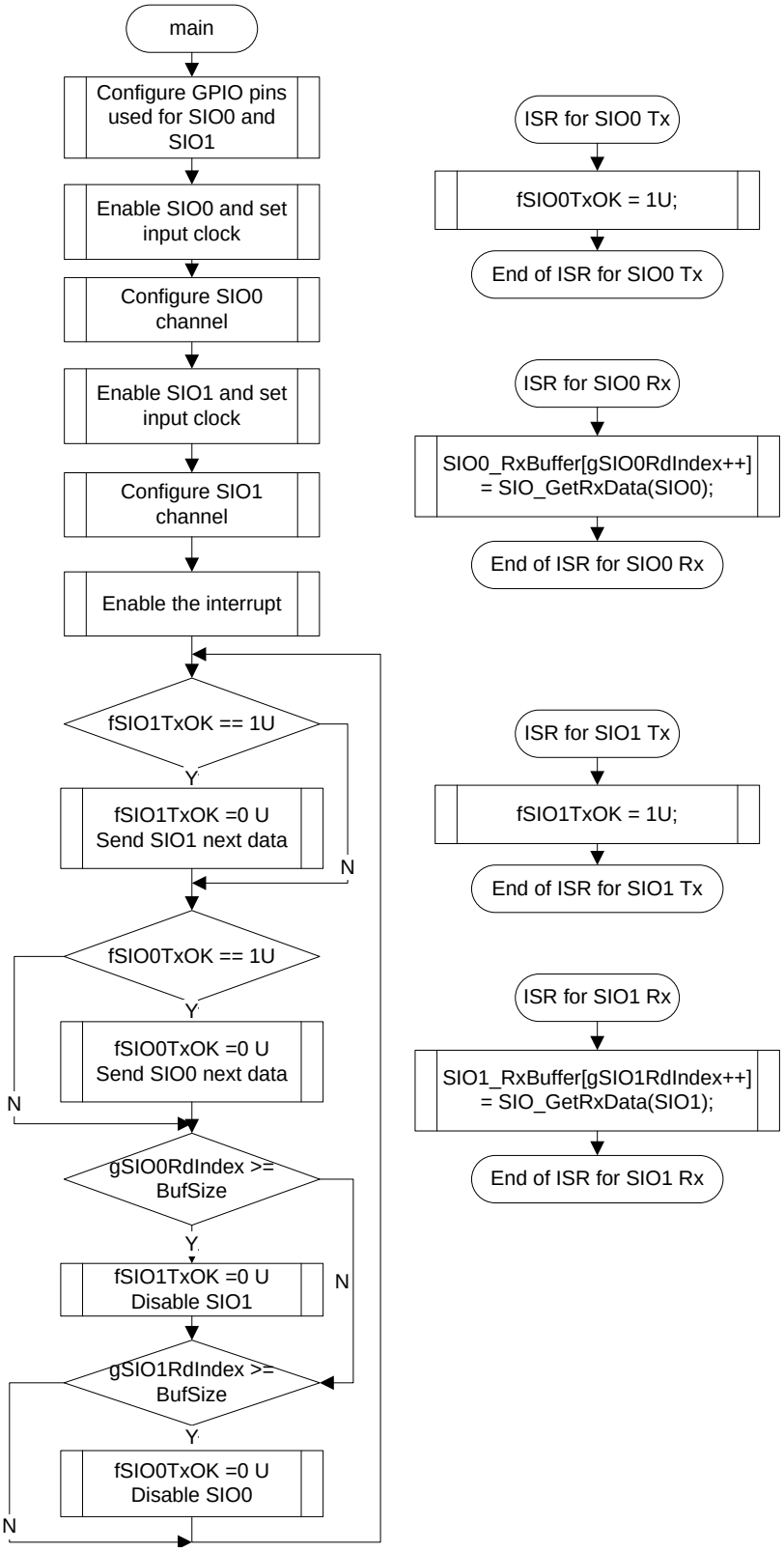
7-11-4 Example: SIO

This is a simple example based on the TX03 Peripheral Driver (SIO).

The example includes:

1. Basic setup operation of SIO
2. The data transfer between SIO0 and SIO1
3. SIO interrupt of Tx and Rx

- **Flowchart**


main
Configure GPIO pins used for SIO0 and SIO1
Enable SIO0 and set input clock
Configure SIO0 channel
Enable SIO1 and set input clock
Configure SIO1 channel
Enable the interrupt
fSIO1TxOK == 1U
Y
fSIO1TxOK =0 U
Send SIO1 next data
N
fSIO0TxOK == 1U
Y
fSIO0TxOK =0 U
Send SIO0 next data
N
gSIO0RdIndex >= BufSize
Y
fSIO1TxOK =0 U
Disable SIO1
N
gSIO1RdIndex >= BufSize
Y
fSIO0TxOK =0 U
Disable SIO0
N
ISR for SIO0 Tx
fSIO0TxOK = 1U;
End of ISR for SIO0 Tx
ISR for SIO0 Rx
SIO0_RxBuffer[gSIO0RdIndex++] = SIO_GetRxData(SIO0);
End of ISR for SIO0 Rx
ISR for SIO1 Tx
fSIO1TxOK = 1U;
End of ISR for SIO1 Tx
ISR for SIO1 Rx
SIO1_RxBuffer[gSIO1RdIndex++] = SIO_GetRxData(SIO1);
End of ISR for SIO1 Rx

- **Code and Explanation for the Example**

At first, configure the GPIO pins for SIO by setting the CR, FR1, IE register of proper port.

Then, enable SIO0 configure the input clock and SIO0 initialization structure.

```
/*Enable the SIO0 channel */
SIO_Enable(SIO0);

/*initialize the SIO0 struct */
SIO0_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO0_Init.IntervalTime = SIO_SINT_TIME_SCLK_8;
SIO0_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO0_Init.TransferDir = SIO_LSB_FRIST;
SIO0_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO0_Init.DoubleBuffer = SIO_WBUF_ENABLE;
SIO0_Init.BaudRateClock = SIO_BR_CLOCK_T4;
SIO0_Init.Divider = SIO_BR_DIVIDER_2;
SIO_Init(SIO0, SIO_CLK_BAUDRATE, &SIO0_Init);
```

Enable SIO1 configure the input clock and SIO1 initialization structure.

```
/*Enable the SIO1 channel */
SIO_Enable(SIO1);

/*initialize the SIO1 struct */
SIO1_Init.InputClkEdge = SIO_SCLKS_TXDF_RXDR;
SIO1_Init.TransferMode = SIO_TRANSFER_FULLDPX;
SIO1_Init.TransferDir = SIO_LSB_FRIST;
SIO1_Init.Mode = SIO_ENABLE_TX | SIO_ENABLE_RX;
SIO1_Init.DoubleBuffer = SIO_WBUF_ENABLE;

SIO_Init(SIO1, SIO_CLK_SCLKINPUT, &SIO1_Init);
```

Enable the interrupt of SIO TX, RX.

```
/* Enable SIO0 Channel TX interrupt */
NVIC_EnableIRQ(INTTX0_IRQn);
/* Enable SIO1 Channel RX interrupt */
NVIC_EnableIRQ(INTRX1_IRQn);

/* Enable SIO1 Channel TX interrupt */
NVIC_EnableIRQ(INTTX1_IRQn);
/* Enable SIO0 Channel RX interrupt */
NVIC_EnableIRQ(INTRX0_IRQn);
```

After all the basic configure, Enter the data transfer routine .

```
while (1) {
        /* SIO1 send data from TXD1*/
        if (fSIO1TxOK == 1U) {
            fSIO1TxOK = 0U;
            SIO_SetTxData(SIO1, SIO1_TxBuffer[gSIO1WrIndex++]);
        } else {
            /*Do Nothing */
        }
        /* SIO0 send data from TXD0*/
        if (fSIO0TxOK == 1U) {
            fSIO0TxOK = 0U;
            SIO_SetTxData(SIO0, SIO0_TxBuffer[gSIO0WrIndex++]);
        } else {
```

```
            /*Do Nothing */
        }

        /*SIO0 receive data end */
        if (gSIO0RdIndex >= BufSize) {
            fSIO1TxOK = 0U;
            SIO_Disable(SIO1);
        } else {
            /*Do Nothing */
        }
        /*SIO1 receive data end */
        if (gSIO1RdIndex >= BufSize) {
            fSIO0TxOK = 0U;
            SIO_Disable(SIO0);
        } else {
            /*Do Nothing */
        }
    }
```

In ISR of SIO0 Tx, set transfer is ok.

```
void INTTX0_IRQHandler (void)
{
    fSIO0TxOK = 1U;
}
```

In ISR of SIO0 Rx, get the receive data from RX buffer

```
void INTRX0_IRQHandler(void)
{
    SIO0_RxBuffer[gSIO0RdIndex++] = SIO_GetRxData(SIO0);
}
```

In ISR of SIO1 Tx, set transfer is ok.

```
void INTTX1_IRQHandler(void)
{
    fSIO1TxOK = 1U;
}
```

In ISR of SIO1 Rx, get the receive data from RX buffer.

```
void INTRX1_IRQHandler(void)
{
    SIO0_RxBuffer[gSIO1RdIndex++] = SIO_GetRxData(SIO1);
}
```

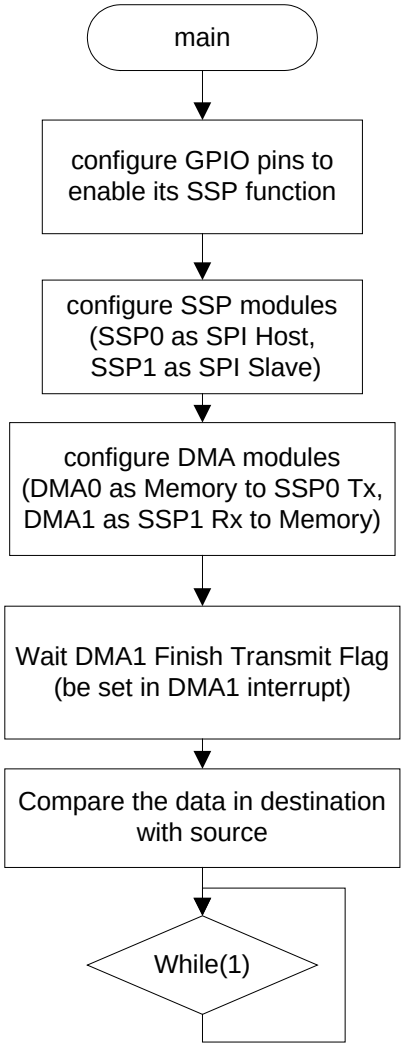
7-12 SSP

7-12-1 Example: SSP0 to SSP1 via DMAC

For M384, there are 2 SSP channels, SSP0 is set as SPI Mast while SSP1 is set as SPI Slave, then connect correspond pins, use DMA to transmit/receive infinite self increased data.

* SSP0 to SSP1 via DMAC demo should run on TOSHIBA TMPM384-SK Board.

- **Flowchart**



- **Code and Explanation for the Example**

Below is the main.c for the flowchart above.

```
int main(void)
{
	GPIO_SetSSP();
	initSSP();
	InitDMA();

	/* Wait the end of transmission */
	while (TxEndFlag != DONE) {
		/* Do nothing */
	}

	/* now DMA is finished, Set a Break Point here, */
	/* after function Buffercompare() is called, result == SAME */
	result = Buffercompare( SRC_Buffer, DST_Buffer, BUFFER_SIZE);

	while(1) {
		/* do nothing */
	}
```

```
}
```

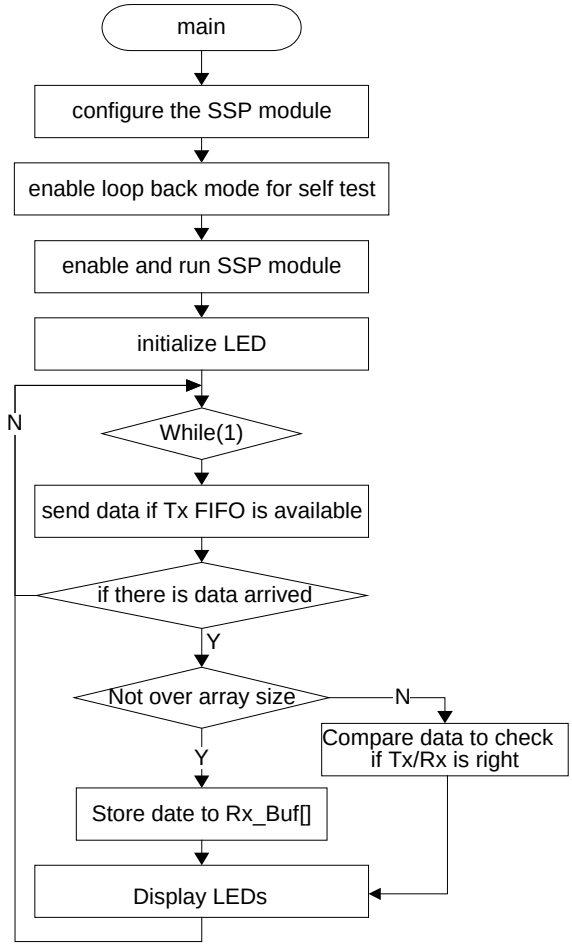
For detail of functions: GPIO_SetSSP(), initSSP(), InitDMA() above, please refer to the comments in the code file.

7-12-2 Example: SSP0 Self Loop Back

This is a simple example based on the TX03 Peripheral Driver (SSP, GPIO).

The example includes:
1. Configuration and initialization SSP module channel 0
2. Enable its loop back mode to do self Tx/Rx

- **Flowchart**



- **Code and Explanation for the Example**

```
/* main function */
int main(void)
{
    SSP_InitTypeDef initSSP;
```

```
    SSP_FIFOState fifoState;

    uint16_t datTx = 0U;            /* must use 16bit type */
    uint32_t cntTx = 0U;
    uint32_t cntRx = 0U;

    uint16_t receive = 0U;
    uint16_t Rx_Buf[MAX_BUFSIZE] = { 0U };
    uint16_t Tx_Buf[MAX_BUFSIZE] = { 0U };

    /* configure the SSP module */
    initSSP.FrameFormat = SSP_FORMAT_SPI;

    /* default is to run at maximum bit rate */
    initSSP.PreScale = 2U;
    initSSP.ClkRate = 1U;
    /* define BITRATE_MIN to run at minimum bit rate */
    /*     BitRate = fSYS / (PreScale x (1 + ClkRate)) */
#ifdef BITRATE_MIN
    initSSP.PreScale = 254U;
    initSSP.ClkRate = 255U;
#endif
    initSSP.ClkPolarity = SSP_POLARITY_LOW;
    initSSP.ClkPhase = SSP_PHASE_FIRST_EDGE;
    initSSP.DataSize = 16U;
    initSSP.Mode = SSP_MASTER;
    SSP_Init(TSB_SSP0, &initSSP);

    /* enable loop back mode for self test */
    SSP_SetLoopBackMode(TSB_SSP0, ENABLE);

    /* enable and run SSP module */
    SSP_Enable(TSB_SSP0);

    /* initialize LEDs on M384-SK board before display something */
    LED_Init();

    while (1) {

        datTx++;
        /* send data if Tx FIFO is available */
        fifoState = SSP_GetFIFOState(TSB_SSP0, SSP_TX);
        if ((fifoState == SSP_FIFO_EMPTY) || (fifoState == SSP_FIFO_NORMAL)) {
            SSP_SetTxData(TSB_SSP0, datTx);
            if (cntTx < MAX_BUFSIZE) {
                Tx_Buf[cntTx] = datTx;
                cntTx++;
            } else {
                /* do nothing */
            }
        } else {
            /* do nothing */
        }

        /* check if there is data arrived */
        fifoState = SSP_GetFIFOState(TSB_SSP0, SSP_RX);
        if ((fifoState == SSP_FIFO_FULL) || (fifoState == SSP_FIFO_NORMAL)) {
            receive = SSP_GetRxData(TSB_SSP0);
            if (cntRx < MAX_BUFSIZE) {
```

```
                    Rx_Buf[cntRx] = receive;
                    cntRx++;
                } else {
                    /* Place a break point here to check if receive data is right.                    */
                    /* Success Criteria:                                                              */
                    /*                    Every data transmited from Tx_Buf is received in Rx_Buf. */
                    /* When the line "#define BITRATE_MIN" is commented, the SSP is run in maxium */
                    /*   bit rate, so we can find there is enough time to transmit date from 1 to   */
                    /*   MAX_BUFSIZE one by one. but if we uncomment that line, SSP is run in */
                    /*   minimum bit rate, we will find that receive data can't catch "datTx++",    */
                    /*   in this so slow bit rate, when the Tx FIFO is available, the cntTx has     */
                    /*   been increased so much.                                                    */
                    __NOP();
                    result = Buffercompare( Tx_Buf, Rx_Buf, MAX_BUFSIZE);

                    if (result == NOT_SAME) {
                        LED_Off(LED1);
                        LED_On(LED0);
                    } else {
                        LED_On(LED1);
                        LED_Off(LED0);
                    }
                }

            } else {
                /* do nothing */
            }

            DisplayLED(receive);
        }
}
```

7-13 TMRB

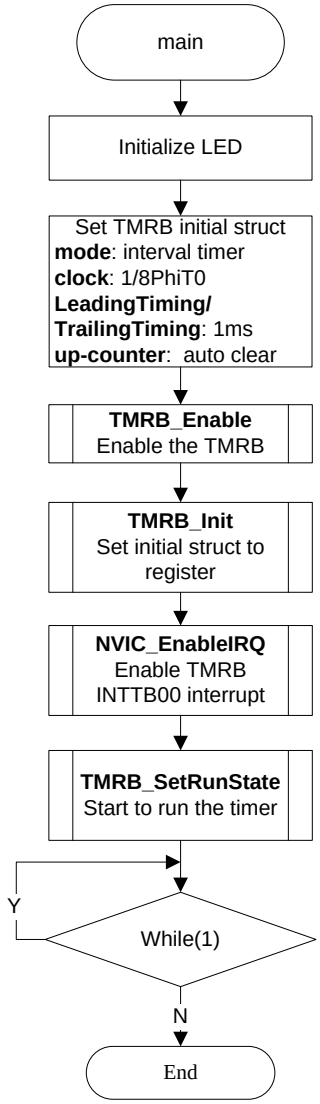
7-13-1 Example: General Timer

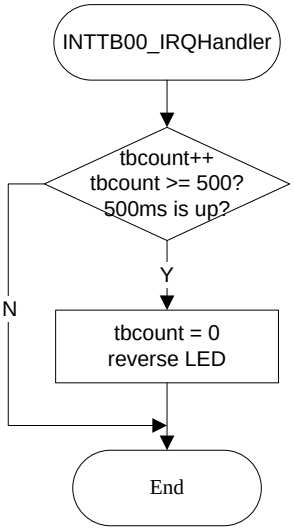
This is a simple example based on the TX03 Peripheral Driver (TMRB, GPIO).

The example includes:

1. TMRB0 initialization
2. General Timer for 1ms

- **Flow Chart**





- **Code and Explanation for the Example**

At first, initialize LED channel on Eval board and turn on LED.

```
LED_Init();                                    /* LED initialize */
LED_On(LED_ALL);                               /* Turn on LED_ALL */
```

Prepare TMRB initialization structure, fill TMRB mode, clock, up-counter clear method, trailing timing and leading timing. This demo will set trailing timing and leading timing to 1ms, macro TMRB_1MS equals 0x1388, because fphiT0 = fsys = fc = 10MHz * PLL = 40MHz, ftmrb = 1/8 fphiT0 = 5MHz, Ttmrb = 0.2us, 1ms/0.2us = 5000 = 0x 1388 (Please see CG part for more detail information about the clock setting)

```
TMRB_InitTypeDef m_tmrb;

m_tmrb.Mode = TMRB_INTERVAL_TIMER;    /* internal timer */
m_tmrb.ClkDiv = TMRB_CLK_DIV_8;       /* 1/8PhiT0 */
m_tmrb.TrailingTiming = TMRB_1MS;     /* periodic time is 1ms */
m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;   /* up-counter auto clear */
m_tmrb.LeadingTiming = TMRB_1MS;      /* periodic time is 1ms */
```

Enable the TMRB module and set the initialization structure to specified registers. Enable INTTB0 interrupt, this interrupt will be triggered every 1ms, in the end, start the TMRB to run.

```
TMRB_Enable(TSB_TB0);                          /* enable the TMRB0 */
TMRB_Init(TSB_TB0, &m_tmrb);                   /* initial the TMRB0 */
NVIC_EnableIRQ(INTTB0_IRQn);                   /* enable INTTB0 interrupt */
TMRB_SetRunState(TSB_TB0, TMRB_RUN);/* run TMRB0*/
```

Then the main routine will enter “While(1)” to wait the interrupt happens. In interrupt routine, use a counter to count. If 500ms is up, reverse the LED and count again.

```
tbcount++;
if (tbcount >= 500U) {                         /* 500ms is up */
    tbcount = 0U;
    /* reverse LED output */
    ledon = (ledon == 0U) ? 1U : 0U;
    if (0U == ledon) {
        LED_Off(LED_ALL);
    } else {
        LED_On(LED_ALL);
    }
} else {
    /* do nothing */
}
```

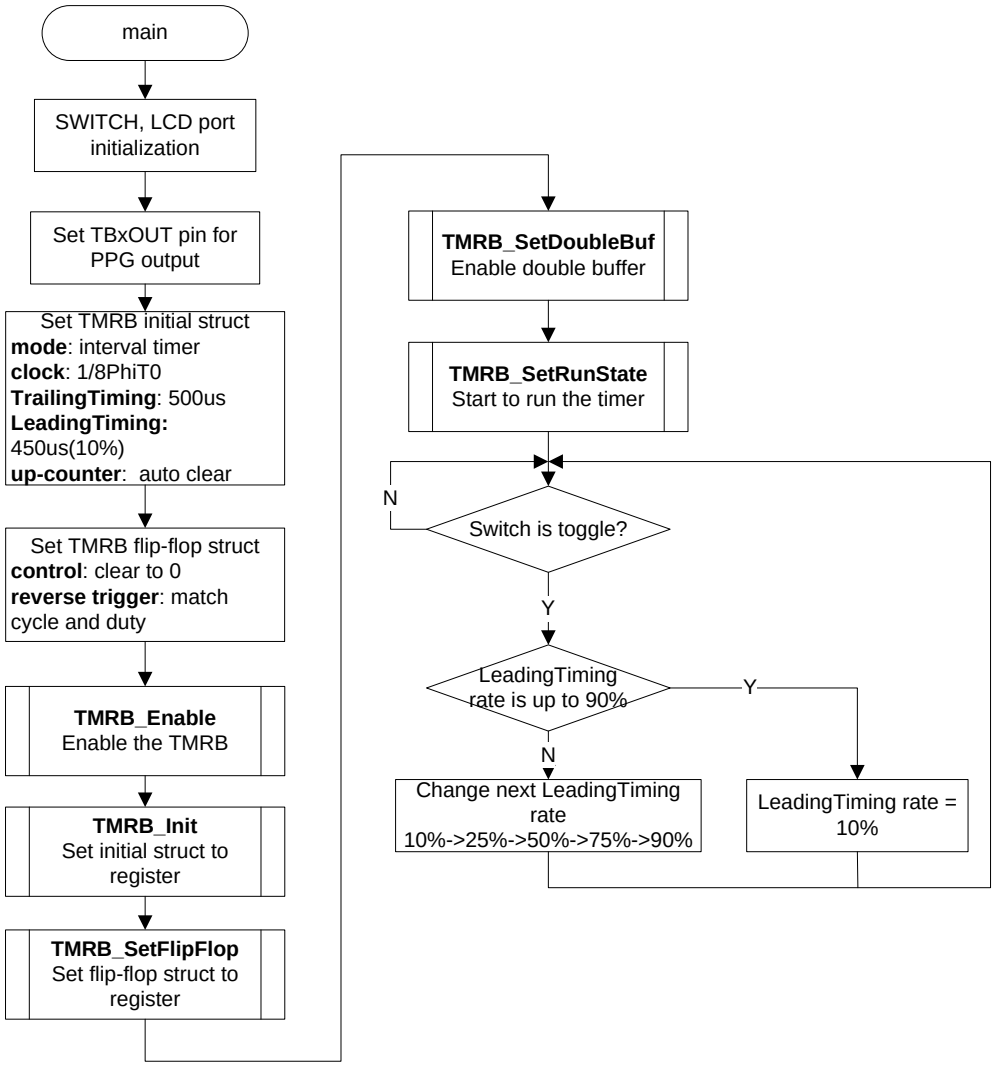
7-13-2 Example: PPG Output

This is a simple example based on the TX03 Peripheral Driver (TMRB, GPIO).

The example includes:

1. TMRB6 initialization
2. PPG process setting and start
3. PPG leading timing adjustment

- **Flow Chart**



- **Code and Explanation for the Example**

At first, initialize SWITCH and LCD channel, set PK1 as TB6OUT for PPG output.

```
/* LCD & switch initialization */
LCD_Configuration();
SW_Init();

/* Set PA5 as TB6OUT for PPG output */
GPIO_SetOutput(GPIO_PA, GPIO_BIT_5);
GPIO_EnableFuncReg(GPIO_PA, GPIO_FUNC_REG_2, GPIO_BIT_5);
```

Prepare TMRB initialization structure, and then fill TMRB mode, clock, up-counter clear method, trailing timing and leading timing into it. This demo will set trailing timing to 500us, macro TMRB6TIME equals 0x09C4, because fphiT0 = fsys = fc = 10MHz * PLL = 40MHz, ftmrb = 1/8 fphiT0 = 5MHz, Ttmrb = 0.2us, 500us/0.2us = 2500 = 0x09C4 (Please see CG part for more detail information about the clock setting)

```
TMRB_InitTypeDef m_tmrb;

m_tmrb.Mode = TMRB_INTERVAL_TIMER;    /* internal timer */
m_tmrb.ClkDiv = TMRB_CLK_DIV_8;       /* 1/8PhiT0 */
```

```
    m_tmrb.TrailingTiming = TMRB6TIME;          /* trailing timing is 500us */
    m_tmrb.UpCntCtrl = TMRB_AUTO_CLEAR;   /* up-counter auto clear */
    m_tmrb.LeadingTiming = LeadingTiming[Rate];     /* leading timing, initial value
10% */
```

Set flip-flop initialization structure, and fill flip-flop control, reverse trigger parameter into it. Reverse trigger is set to match leading timing and match trailing timing.

```
    PPGFFInital.FlipflopCtrl = TMRB_FLIPFLOP_CLEAR;
    PPGFFInital.FlipflopReverseTrg=TMRB_FLIPFLOP_MATCH_TRAILINGTIMING|
    TMRB_FLIPFLOP_MATCH_LEADINGTIMING;
```

Enable the TMRB module and set the initialization structure and flip-flop structure to specified registers. Enable double buffer, and disable capture function. In the end, start the TMRB to run.

```
    TMRB_Enable(TSB_TB6);
    TMRB_Init(TSB_TB6, &m_tmrb);
    TMRB_SetFlipFlop(TSB_TB6, &PPGFFInital);
    /* enable double buffer */
    TMRB_SetDoubleBuf(TSB_TB6,ENABLE, TMRB_WRITE_REG_SEPARATE);
    TMRB_SetRunState(TSB_TB6, TMRB_RUN);
```

Wait switch from low to high, and at the same time, display current leading timing on LED.

```
    do {      /* wait if switch is Low */
              keyvalue = GPIO_ReadDataBit(KEYPORT, GPIO_BIT_0);
              LeadingTiming_display(); /* display current leading timing */
       } while (GPIO_BIT_VALUE_0 == keyvalue);
```

If the switch is changed to high, change the leading timing according to 10%->25%->50%->75%->90%, then from 90% to 10% again.

```
    Rate++;
    if (Rate >= LEADINGTIMINGMAX) {
         Rate = LEADINGTIMINGINIT;
         } else {
              /* Do nothing */
         }

    TMRB_ChangeLeadingTiming(TSB_TB6,   LeadingTiming[Rate]);       /*  change
leading timing rate */
```

The calculation method of leading timing value:
TrailingTiming = 500us, ftmrb = 1/8 fphiT0 = 5MHz, Ttmrb = 0.2us (these time parameters will be different if use different CG setting)

LeadingTiming = 10%: high-level time is 500*10% = 50us, low-level time is 500-50 = 450us, counter value = 450us/Ttmrb = 0x8CA

LeadingTiming = 25%: high-level time is 500*25% = 125us, low-level time is 500-125 = 375us, counter value = 375us/Ttmrb = 0x753

LeadingTiming = 50%: high-level time is 500*50% = 250us, low-level time is 500-250 = 250us, counter value = 250us/Ttmrb = 0x4E2

LeadingTiming = 75%: high-level time is 500*75% = 375us, low-level time is 500-375 = 125us, counter value = 125us/Ttmrb = 0x271

LeadingTiming = 90%: high-level time is 500*90% = 450us, low-level time is 500-450 = 50us, counter value = 50us/Ttmrb = 0xFA

That is the calculation method of leading timing array

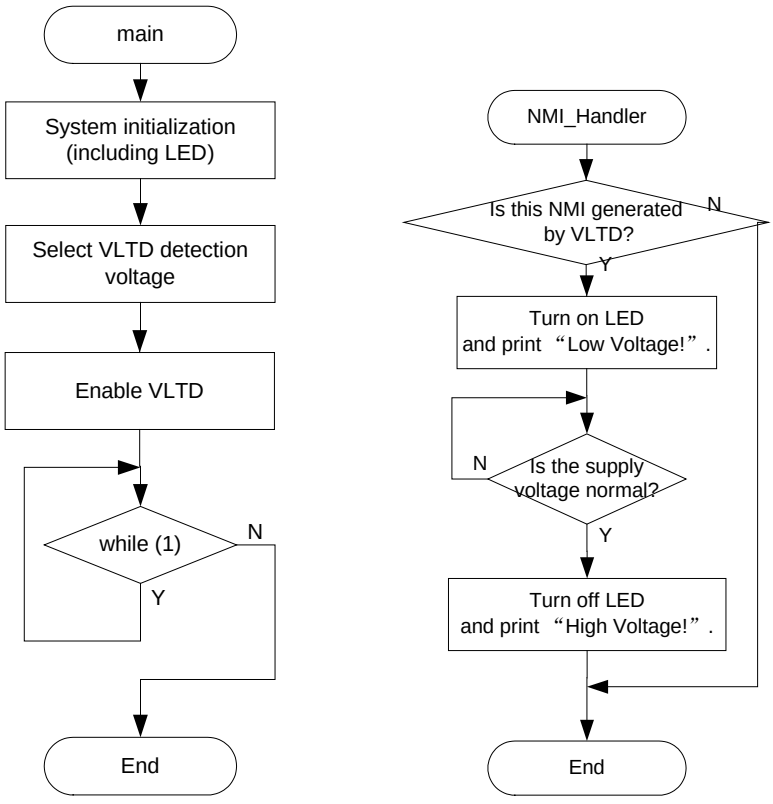
```
uint32_t LeadingTiming[5] = { 0x8CAU, 0x753U, 0x4E2U, 0x271U, 0xFAU };
/* leading timing: 10%, 25%, 50%, 75%, 90% */
```

7-14 VLTD

This is a simple example based on the TX03 Peripheral Driver (VLTD, GPIO).

The example includes:
1. VLTD configuration
2. VLTD status monitoring

- **Flowchart**



- **Code and Explanation for the Example**

Main program : [main.c]

In void main(void):

Initialize the system first. Disabe WDT explicitly to prevent confusion.

```
WDT_Disable ();
LEDInit();
```

Set detection voltage and enable VLTD.

```
VLTD_SetVoltage(VLTD_DETECT_VOLTAGE_46);
VLTD_Enable ();
```

Then a usual while(1) loop.

NMI handler : [tmpm384_vltd_int.c]

In void NMI_Handler(void):

Make sure the NMI is generated by VLTD.

```
if (CG_GetNMIFlag().Bit.VoltageDetection == 1U) {
{
```

Turn on LED and print a message to the terminal.

```
        LedOn(LED3);
#ifdef DEBUG
        printf("Low voltage!\n");
#endif
```

Wait for the power supply voltage to be normal.

```
        while (VLTD_GetStatus() == 1U)
        {
            __NOP();
        }
```

Turn off LED and print a message to the terminal.

```
        LedOff(LED3);
#ifdef DEBUG
        printf("High voltage!\n");
#endif
}
```

At last the NMI_Handler() function returns.

7-15 WDT

This is a simple example based on the TX03 Peripheral Driver (WDT, GPIO).

The example includes:

1. WDT initialization.
2. In DEMO1 WDT isn't cleared before timer overflow, NMI interrupt is generated.
3. In DEMO2 WDT is cleared before timer overflow, the LED will blink all the time.

- **Code and Explanation for the Example**

The following code is an example for initializing WDT. Detect time is set to 2^25/fsys, and interrupt will be generated when timer overflow.

```
WDT_InitTypeDef WDT_InitStruct;
```

```
WDT_InitStruct.DetectTime = WDT_DETECT_TIME_EXP_25;
WDT_InitStruct.OverflowOutput = WDT_NMIINT;
```

Initialize WDT and enable it.

```
WDT_Init(&WDT_InitStruct);
WDT_Enable();
```

In DEMO1 Waiting for the NMI interrupt flag.

```
while(1)
{
}
```

In DEMO1 When NMI interrupt is occurred the WDT will be disabled and the LED blink will be stopped.

```
WDT_Disable();
```

In DEMO2 WDT will be cleared and the LED will blink all the time.

```
WDT_WriteClearCode();
```

7-16 ENC

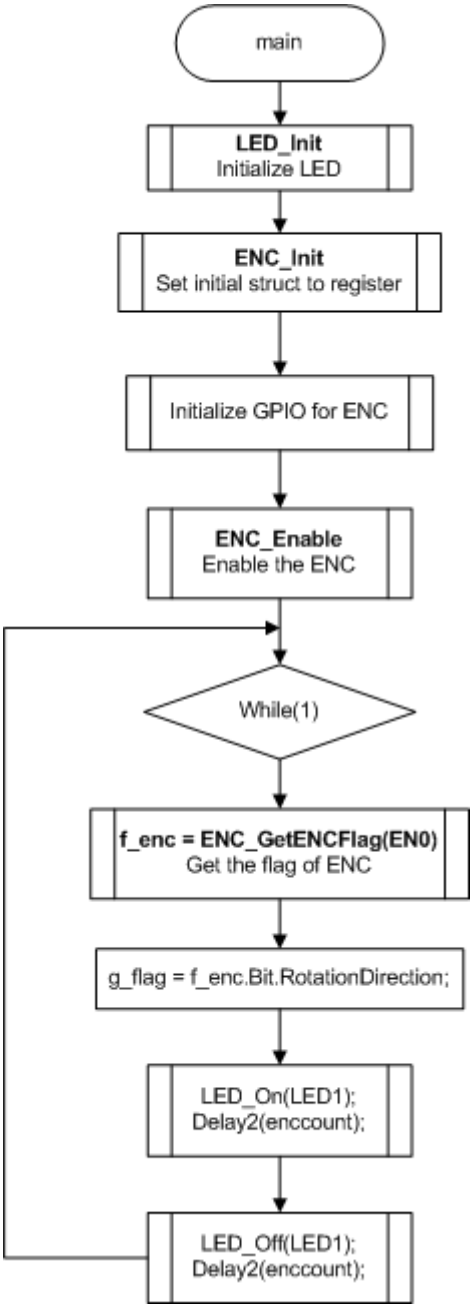
7-16-1 Example: Rolling Detection

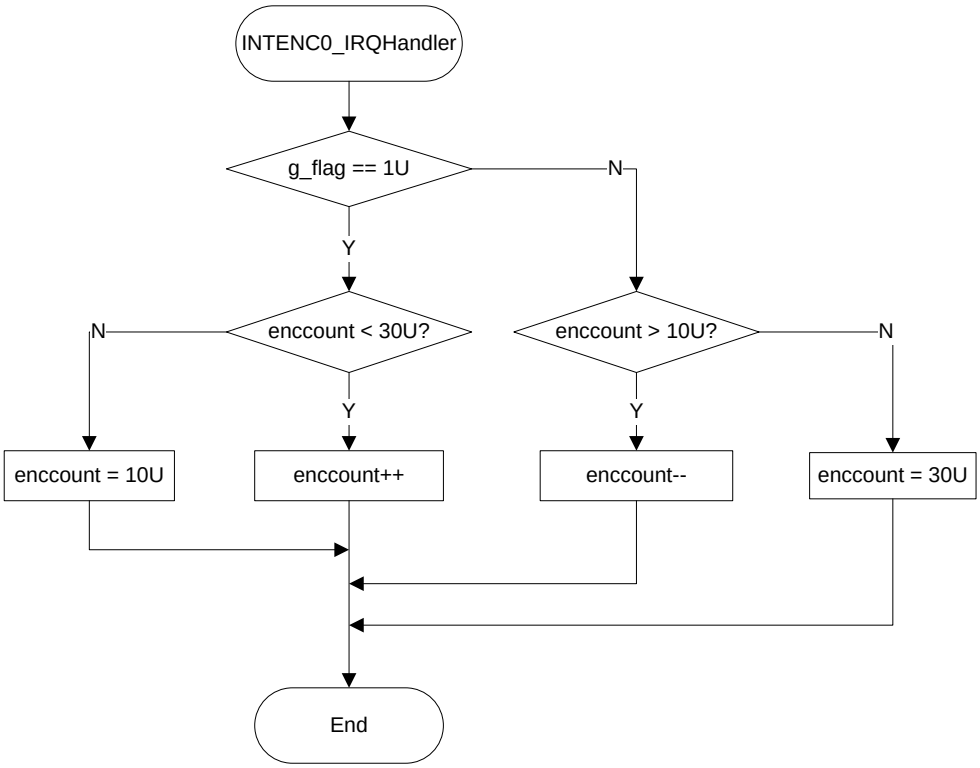
This is a simple example based on the TX03 Peripheral Driver (ENC, GPIO).

The example includes:

1. ENC0 initialization.
2. Rolling detection of mouse wheel.

- **Flowchart**


main
LED_Init
Initialize LED
ENC_Init
Set initial struct to register
Initialize GPIO for ENC
ENC_Enable
Enable the ENC
While(1)
f_enc = ENC_GetENCFlag(EN0)
Get the flag of ENC
g_flag = f_enc.Bit.RotationDirection;
LED_On(LED1);
Delay2(enccount);
LED_Off(LED1);
Delay2(enccount);



- **Code and Explanation for the Example**

The following simple example is based on TX03 Peripheral Driver (ENC), which will detect the rolling of mouse wheel.

At first, initialize LED channel on eval board.

```
/* LED initialization */
LED_Init();
```

Prepare ENC initialization structure, and initialize the ENC0.

```
/* ENC0 initialization */
m_enc.ModeType = ENC_ENCODER_MODE;
m_enc.PhaseType = ENC_TWO_PHASE;
m_enc.CompareStatus = ENC_COMPARE_DISABLE;
m_enc.ZphaseStatus = ENC_ZPHASE_DISABLE;
m_enc.FilterValue = ENC_FILTER_VALUE31;
m_enc.IntEn = ENC_INTERRUPT_ENABLE;
m_enc.PulseDivFactor = ENC_PULSE_DIV1;

ENC_Init(EN0,&m_enc);
ENC_SetCounterReload(EN0,0xFFFU);
```

Initialize the GPIO for ENC0.Set the PD0 and PD1 as inputs of ENC0.

```
/* GPIO initialization */
GPIO_SetInputEnableReg(GPIO_PD, GPIO_BIT_0, ENABLE);/*Set PD0 as input */
GPIO_EnableFuncReg(GPIO_PD, GPIO_FUNC_REG_1, GPIO_BIT_0);/*Set PD0 as ENCA0 */
GPIO_SetInputEnableReg(GPIO_PD, GPIO_BIT_1, ENABLE);/*Set PD1 as input */
```

```
    GPIO_EnableFuncReg(GPIO_PD, GPIO_FUNC_REG_1, GPIO_BIT_1);/*Set
PD1 as ENCB0 */
```

Enable the ENC0 and ENC0 interrupt.

```
    /* Enable ENC0 */
    ENC_Enable(EN0);
    /* Enable ENC0 interrupt */
    NVIC_EnableIRQ(INTENC0_IRQn);
```

Get the status of rotation direction of ENC0 to indicate the rolling direction of the mouse wheel.Blink the LED1 according to the cycle of rolling

```
    /* LED1 blink speed based on the rolling of mouse wheel */
    while(1){
        f_enc = ENC_GetENCFlag(EN0);
        g_flag = f_enc.Bit.RotationDirection;
        LED_On(LED1);
        Delay2(enccount);
        LED_Off(LED1);
        Delay2(enccount);
    }
```

Enccount is the count of rolling, and it changed in the interupt routine.
While rolling the mouse wheel forward, enccount decreased. When enccount reaches to 10, it will back to 30.
While rolling the mouse wheel backward, enccount added. When enccount reaches to 30, it will back to 10.

```
   if(g_flag == 1U){
        if(enccount < 30U){
            enccount++;
        } else {
            enccount = 10U;
        }
    } else {
        if(enccount > 10U){
            enccount--;
        } else {
            enccount = 30U;
        }
    }
```

7-17 PMD

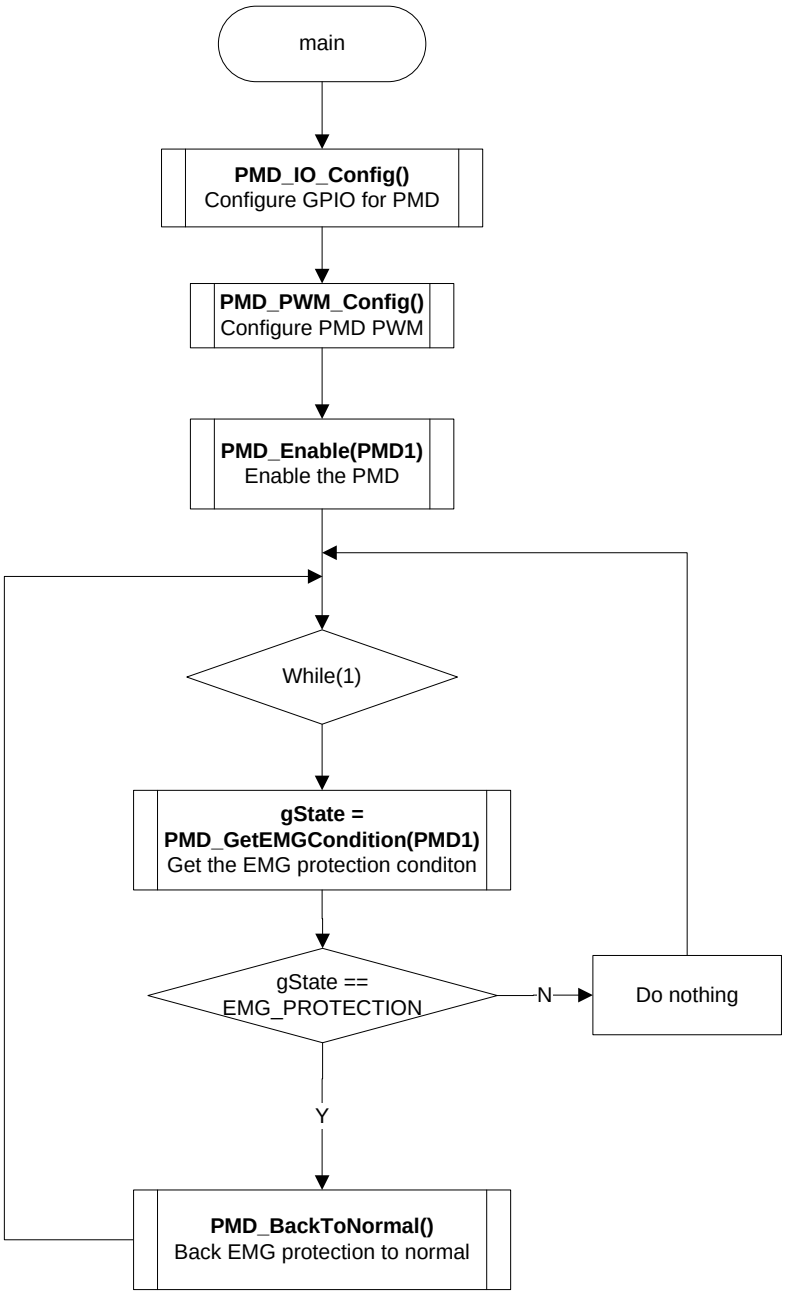
7-17-1 Example: Phase Output

This is a simple example based on the TX03 Peripheral Driver (PMD, GPIO).

The example includes:

1. PMD and GPIO initialization.
2. EMG protection detection.
3. Back EMG protection to normal.

- **Flowchart**



- **Code and Explanation for the Example**

At first, configure the LED and PMD and GPIO.

```
/* LED initialization */
LED_Init();

/* GPIO configuration*/
PMD_IO_Config();

/* PMD PWM configuration*/
PMD_PWM_Config();
```

If the DEMO_U_PHASE is defined, the duty mode is U-phase in common,
If the DEMO_3_PHASE is defined, the duty mode is 3-phase independent.

```
/* PMD1 initialization */
    m_pmd.CycleMode = PMD_PWM_NORMAL_CYCLE;
#ifdef DEMO_U_PHASE
    m_pmd.DutyMode = PMD_DUTY_MODE_U_PHASE;          /* U-phase in common */
#endif
#ifdef DEMO_3_PHASE
    m_pmd.DutyMode = PMD_DUTY_MODE_3_PHASE;          /* 3-phase independent */
#endif
    m_pmd.IntTiming = PMD_PWM_INT_TIMING_MINIMUM;
    m_pmd.IntCycle = PMD_PWM_INT_CYCLE_1;
    m_pmd.CarrierMode = PMD_CARRIER_WAVE_MODE_1;    /* PWM mode 1 (center PWM, triangle wave) */
    m_pmd.CycleTiming = 0x3FFFU;

    PMD_Init(PMD1, &m_pmd);
```

Set the different compare value in the 3 phases.
In the duty mode U-phase in common, the outputs are same according to phase U.
In the duty mode 3-phase independent, the outputs are independent according to the compare value.

```
    PMD_SetAllPhaseCompareValue(PMD1,0x0FFFU,0x1FFFU,0x2FFFU);
```

And then enable the PMD.

```
    PMD_Enable(PMD1);
```

Judge the EMG protection condition.If the condition is in protection condition, LED1 will light on, and then back EMG protection to normal. If the condition is not in protection condition, LED1 will light off.

```
    while(1){
        /* Get the EMG protection condition*/
        gState = PMD_GetEMGCondition(PMD1);
        /* Judge the EMG protection condition */
        if (gState == EMG_PROTECTION) {
            /* LED1 will light on if the condition is in protection */
            LED_On(LED1);
            /* Back EMG protection to normal */
            PMD_BackToNormal();
            Delay(1000U);
        } else {
            /* LED1 will light off if the condition is not in protection */
            LED_Off(LED1);
        }
    }
```